EPSON
EXCEED YOUR VISION

TM-S1000 for EMEA

API Reference Guide

Rev.D
M00008903

Cautions

- ❏ No part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Seiko Epson Corporation.
- ❏ The contents of this document are subject to change without notice. Please contact us for the latest information.
- ❏ While every precaution has taken in the preparation of this document, Seiko Epson Corporation assumes no responsibility for errors or omissions.
- ❏ Neither is any liability assumed for damages resulting from the use of the information contained herein.
- ❏ Neither Seiko Epson Corporation nor its affiliates shall be liable to the purchaser of this product or third parties for damages, losses, costs, or expenses incurred by the purchaser or third parties as a result of: accident, misuse, or abuse of this product or unauthorized modifications, repairs, or alterations to this product, or (excluding the U.S.) failure to strictly comply with Seiko Epson Corporation's operating and maintenance instructions.
- ❏ Seiko Epson Corporation shall not be liable against any damages or problems arising from the use of any options or any consumable products other than those designated as Original EPSON Products or EPSON Approved Products by Seiko Epson Corporation.

Trademarks

EPSON is a registered trademarks of Seiko Epson Corporation in Japan and other countries/regions.

EPSON TM-S1000 Driver is based in part on the work of the Independent JPEG Group.

libtiff

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that (i) the above copyright notices and this permission notice appear in all copies of the software and related documentation, and (ii) the names of Sam Leffler and Silicon Graphics may not be used in any advertising or publicity relating to the software without the specific, prior written permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Celeron®and Pentium® are registered trademarks of Intel Corporation.

MS-DOS®, Microsoft ®, Win32®, Windows®, Windows NT®, Windows Vista®, Windows Server®, Visual Studio®, Visual Basic®, Visual C++ ® and Visual C# ® are trademarks or registered trademarks of Microsoft Corporation in the US or other countries.

InstallShield® is trademarks or registered trademarks of Macrovision Corporation in the US or other countries.

PC/AT® is trademarks of International Business Machines Corporation in the United States.

General Notice: Other product and company names used herein are for identification purposes only and may be trademarks of their respective companies.

Contents

Chapter 1
TM-S1000 Driver Overview

Introduction

This chapter describes the necessary information when you develop applications using the TM-S1000 API. Basic operations such as magnetic character reading or scanning on sample programs attached to the API and how to program error handling when a paper jam or double feeding occurs are also described.

Contents

Functions of theTM-S1000

The TM-S1000 is a compact image scanner that continuously processes checks or documents. You can realize the following functions by using the scanner with the TM-S1000 API.

- ❏ Reading magnetic characters on a check (E13B, CMC7)
- ❏ Scanning both sides of a document (black and white/256 grayscale).
- ❏ Saving a scanned image in a specified resolution and format (Black and White : TIFF *, BMP / Grayscale : TIFF, JPEG, BMP, Raster).
- ❏ Reading OCR A/B fonts
- ❏ Decoding a barcode data from scanned image data
- ❏ Adding a processing record to front or back image data (Electric endorsement).
- ❏ Franking on a processed document
- ❏ Detecting/Notifying a document double-feeding and paper jams
- ❏ Performing IQA(Image Quality Assurance) analysis of image data. IQA conforms to the recommendations of FSTC (Financial Services Technology Consortium).
- ❏ Sorting documents into 2 pockets depending on the reading result.
- ❏ Notifying in advance that a pocket will be full with the pocket near full sensor.
- ❏ Selecting a process mode.
  - Reading during continuous paper feeding.
  - Feeding a paper after reading it.
  - Processing consecutively for both main and sub pockets (Waterfall function).
- ❏ Obtaining information from the maintenance counter (operation times of each mechanism)
- ❏ Notification of each operation by the embedded buzzer

* Image file that conforms to the ANSI X9.100-181-2007 standard must be saved as black and white, TIFF format, CCITT(Goup 4) compressed, resolution of 200 dpi.

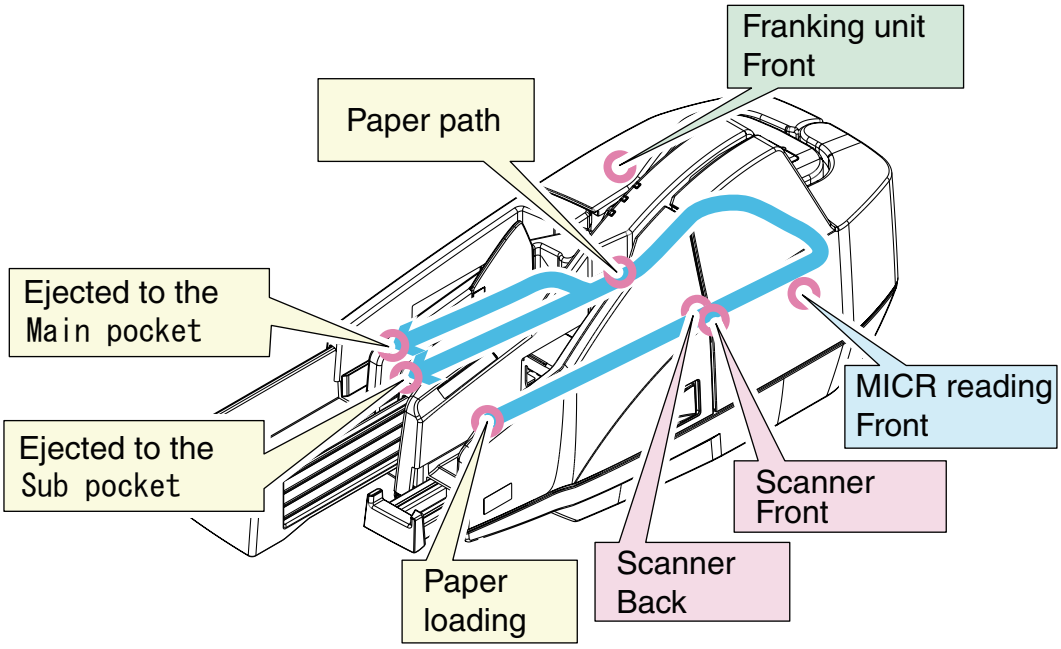
Model

- ❏ 30 dpm model
- ❏ 60 dpm model
- ❏ 90 dpm model

Note:
Waterfall function and BiSetNumberOfDocument is not available for the 30 dpm/60 dpm model with a firmware version of 1.03 or earlier.

Structure of TM-S1000

The main functions of the TM-S1000 are arranged as shown in the figure below. All the functions within the flow from loading to ejecting documents are shown.



Features of the TM-S1000 API

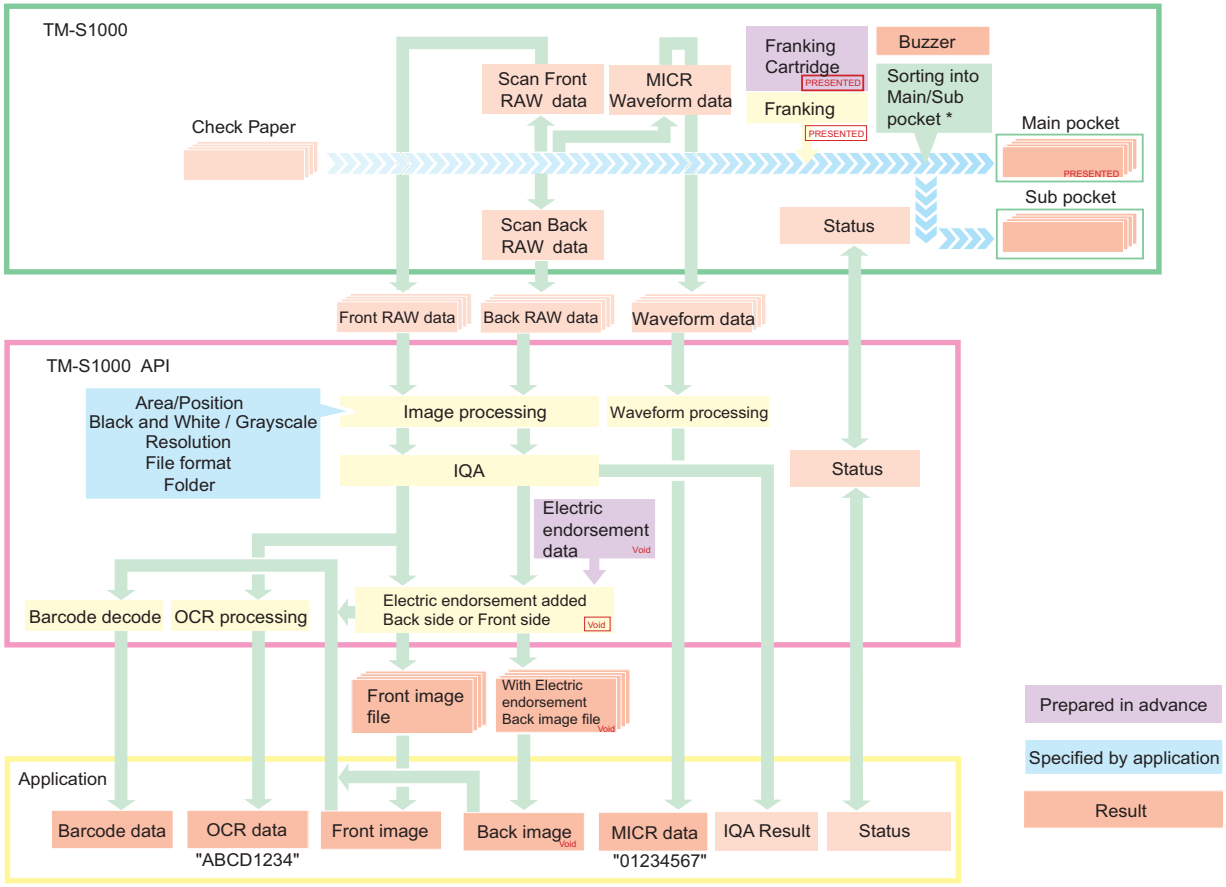
❏ Applications for the TM-J9000/J9100 can be used with a minimum of modification.

- Reimplement modifications referring to Chapter 5.
- Printing and Photo ID functions of the TM-J9000/J9100 are not available for the TM-S1000, which is a device for image scanning only.

❏ Silent installation
Building the API in an application installer enables the API to be installed automatically when you install the application.

❏ Log function
Log files of API used by applications are helpful for troubleshooting.

The figure below shows the flow of document reading, data processing, and functions. Specifying on an application can obtain various results.



* When one pocket becomes full in the Waterfall mode, documents are ejected to another pocket.

TM-S1000 API allows you to make settings for the following TM-S1000 functions.

Functions	Side		Specify/Prepare	Result
	Front	Back		
Scanner reading	✔	✔	Reading area; resolution; Image quality; IQAvalidation; file name; folder name; file format	Image file
MICR reading	✔	-	Reading font	MICR text data
OCR reading	✔	-	Reading area; reading ON/OFF	OCR text data in a specified area
Barcode decode	✔	✔	Specify conditions of decode	Barcode data
Electric endorsement added	✔	✔	Data added ON/OFF; Prepare endorsement data; Rotation angle of endorsement data	Front/Back image data with endorsement data.
Franking	✔	-	Franking ON/OFF; Prepare a franking cartridge	Franking on the front of documents
Document sorting	-		Specify conditions of sorting	Sorting into the main/sub pocket.
Buzzer	-		Specify conditions of buzzer	The buzzer sounds /does not sound
Process mode	-		Specify continuous/individual/ Waterfall	Operating in the specified process mode.

Operating Environment

OS

- ❏ Microsoft Windows 8 (32/64 bit)
- ❏ Microsoft Windows 7 SP1 (32/64 bit)
- ❏ Microsoft Windows Vista SP2 (32/64 bit)
- ❏ Microsoft Windows XP SP3 (32 bit)
- ❏ Microsoft Windows 2000 SP4
- ❏ Microsoft Windows Server 2012
- ❏ Microsoft Windows Server 2008 R2 SP1
- ❏ Microsoft Windows Server 2008 SP2 (32/64 bit)

Computer

30 dpm model/ 60 dpm model

CPU: Pentium 4 1.2 GHz or more

Memory: 256 MB or more than the minimum system requirements of the OS

60 dpm model (when using IQA or Barcode)/ 90 dpm model

CPU: Pentium 4 2.0 GHz or more

Memory: 512 MB or more

Interface

USB2.0

Development Language

Development Tool	Development Language
Visual Studio 6.0	Visual Basic 6.0
	Visual C++ 6.0
Visual Studio .NET 2003	Visual Basic .NET 2003
	Visual C++ .NET 2003
	Visual C# .NET 2003
Visual Studio 2005	Visual Basic 2005
	Visual C++ 2005
	Visual C# 2005
Visual Studio 2008	Visual Basic 2008
	Visual C++ 2008
	Visual C# 2008
Visual Studio 2010	Visual Basic 2010
	Visual C++ 2010
	Visual C# 2010

For development information on .NET, refer to "TM-S1000 .NET API Reference Guide".

Sample programs for 8 levels for each development language are attached to the TM-S1000 API. They make it easy to develop applications.

Files Provided by the TM-S1000 API

When you install the TM-S1000 API, the files are copied into the following folder by default.

C:/Program Files
- EPSON
  - BankDriver
    - TM-S1000
      - bin (DLL necessary for the TM-S1000)
      - TMUSB (USB port driver)

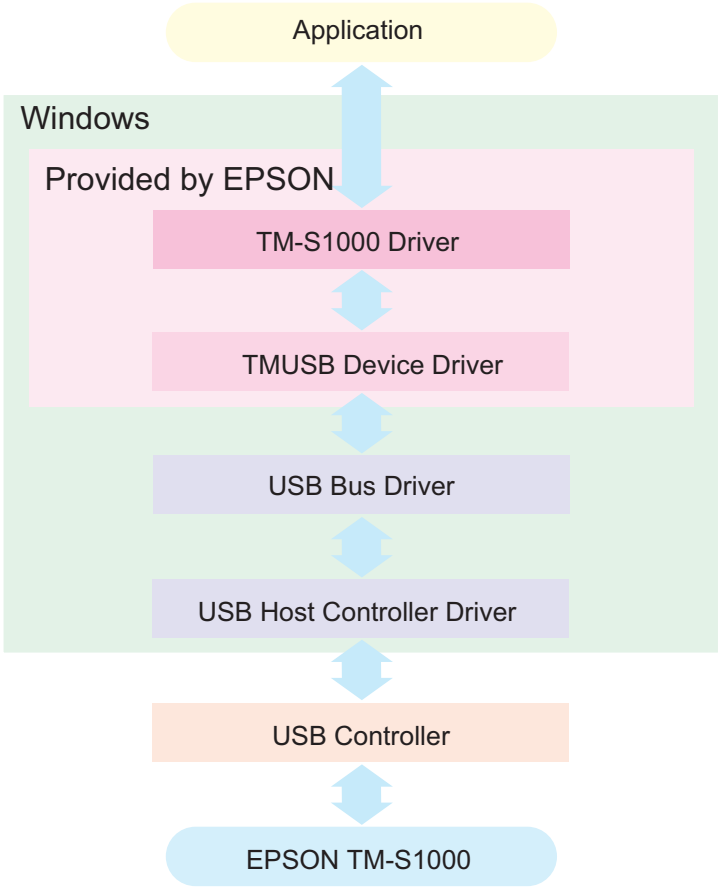
Header file(EpsStmApiInterface.h/MultiFunction.h) included in the sample program.

Note

For 64-bit OS, "Program Files(x86)" corresponds to the above "Program Files" folder.

Roles of Driver

An application controls the TM-S1000 with the API and drivers. The roles are shown below.



Chapter 2
Install and Uninstall

This chapter explains the installation and uninstallation of the TM-S1000 driver. Also explained is the "Silent installation" of the TM-S1000 driver that occurs automatically when the application is installed. Note that installing the TM-S1000 drivers also causes the TM-S1000 API and USB device drivers (TMUSB) to be installed.

Install and Uninstall

This section explains how to install and uninstall the TM-S1000 API.

Note

When updating TM-S1000 driver, install the new driver without uninstalling the existing old driver. The old driver is automatically uninstalled. You also do not need to uninstall the existing driver even when downgrading the driver.

If using .NET application, it is necessary for the .NET Framework to be installed before installing the EPSON TM-S1000 Driver. See the Microsoft website for information concerning the .NET Framework.

Install

The following procedure is used to install the TM-S1000 API.

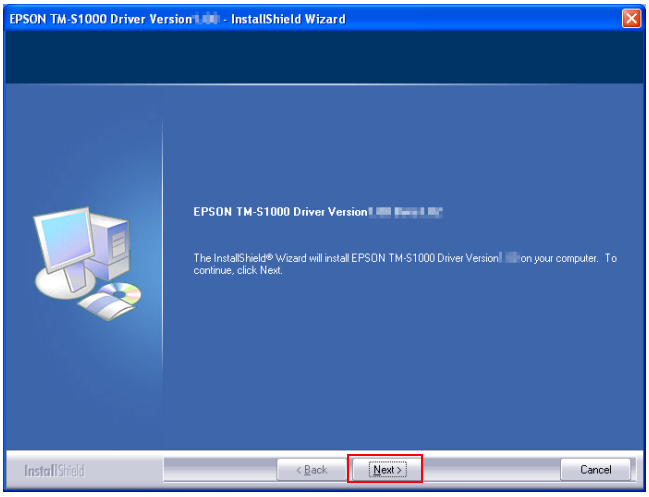
1. Before installing the driver, check that the TM-S1000 is not connected to the computer.
2. Execute setup.exe for the EPSON TM-S1000 driver. 
   When installing the TM-S1000 API under Windows Vista or newer Windows versions, executing setup.exe for the EPSON TM-S1000 driver will cause the User Account Control screen to appear. Note also that the operation varies between those users with Administrator authority and those without.

Users with Administrator authority	:	Click [Allow]. The installation screen will appear.
Users without Administrator authority	:	Input the password of a user with Administrator authority, and then click [OK]. The installation screen will appear.

   Note

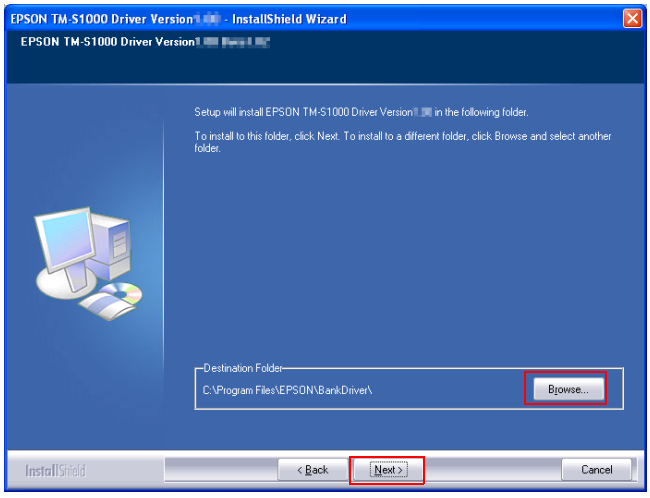
   Do not attempt to execute setup.exe for the EPSON TM-S1000 driver while the TM-S1000 API is being installed.

3. Click [Next].



4. The SOFTWARE LICENSE AGREEMENT screen will appear. Select “I accept the terms of the license agreement” and then click [Next].

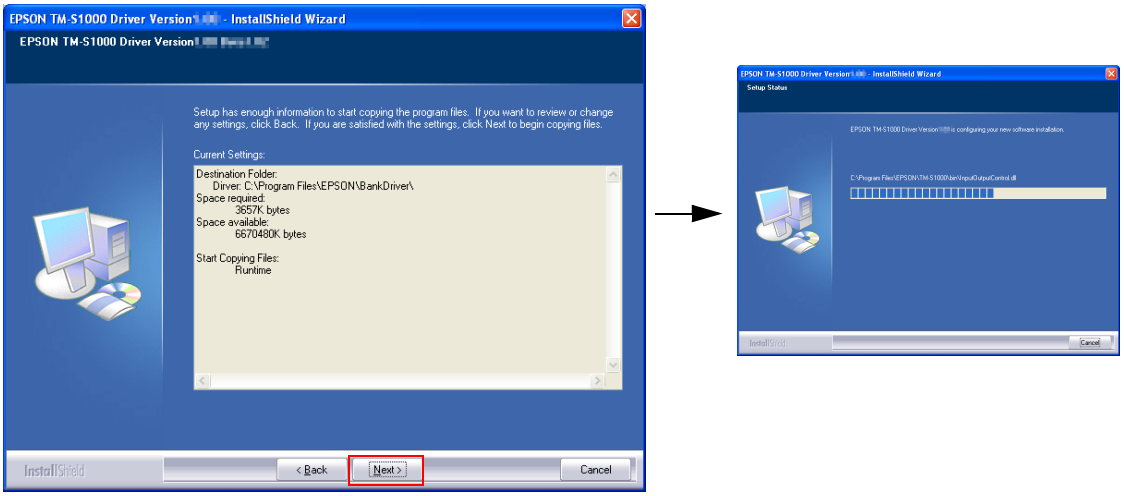
5. Specify the location where the TM-S1000 API is to be installed, and then click the [Next] button. (The default installation location is C:\Program Files\EPSON\BankDriver.)



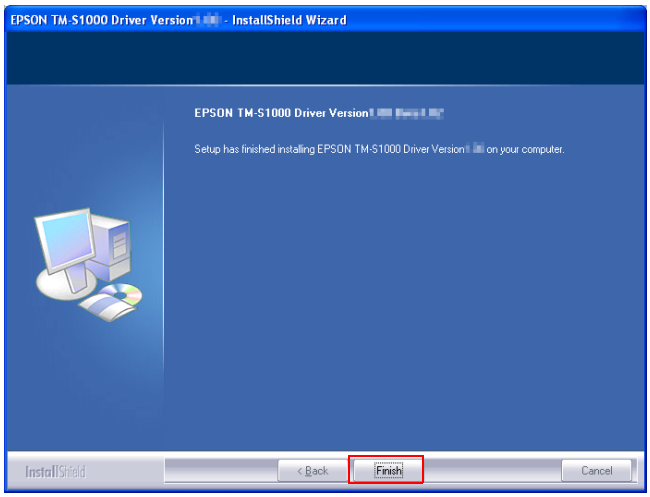
Note

For 64-bit OS, "Program Files(x86)" corresponds to the above "Program Files" folder.

6. The Current Settings screen appears. Click [Next]. The TM-S1000 API is installed.



7. The Maintenance Complete screen appears. Click [Finish]. This completes the installation of the TM-S1000 API.

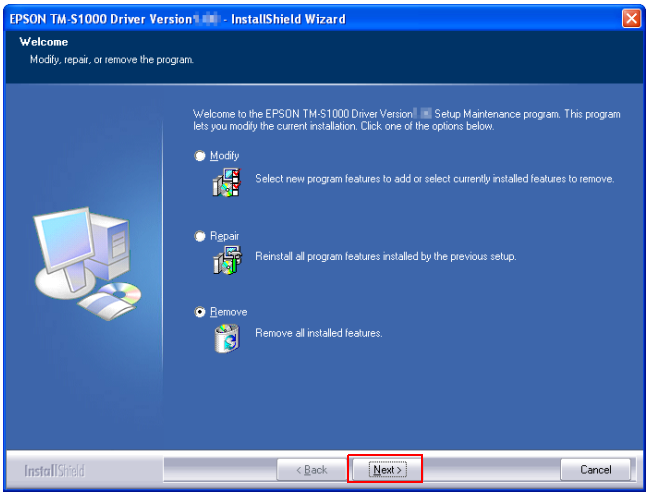


8. Connect the TM-S1000 to your computer and then apply the power.

Uninstall

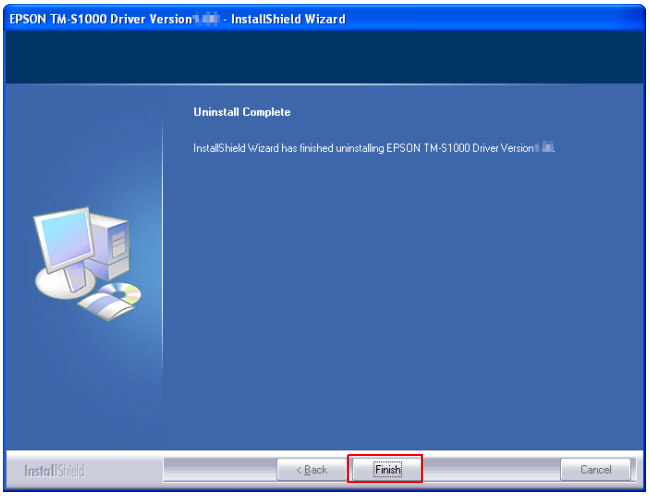
The following procedure is used to uninstall the TM-S1000 API.

1. Execute setup.exe for the EPSON TM-S1000 driver.
2. Select [Remove] and then click [Next].



3. The Uninstall Confirmation screen appears. Click [Yes].

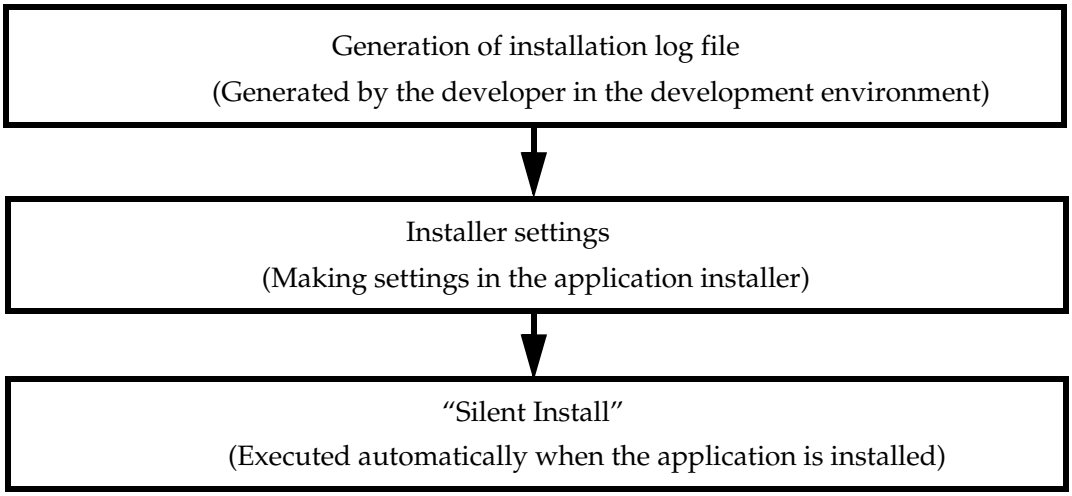
4. The Uninstall Complete screen appears. Click [Finish]. This completes the uninstallation of the TM-S1000 API.



Subsequently, the TM-S1000 API cannot be used. If you subsequently wish to use the TM-S1000 API, it must be reinstalled.

Silent Install

"Silent Install" refers to the automatic installation of the TM-S1000 driver when the user installs an application.
This is done by executing a command and incorporating setup.exe of the EPSON TM-S1000 Driver and the log file recorded during the TM-S1000 driver installation procedure into the installer of the application.



Note

When performing the silent install on a client computer, you must have logged on to the computer as an Administrator. The installation cannot be made if you have logged on as a user.

Generation of installation log file

To create an installation log file, perform installation by executing setup.exe for the TM-S1000 driver from the command prompt.

Note

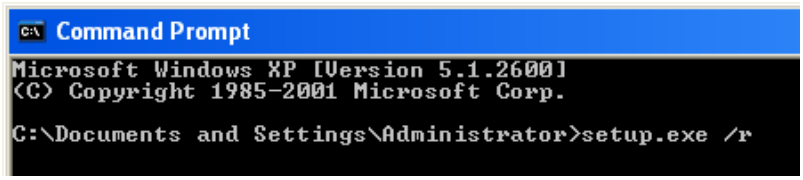
Create the installation log file using a development environment on a computer on which the TM-S1000 API has not been installed.
Before creating the installation log file, first create the TM-S1000 API installation folder appropriate for the application.

Create the log file using the following procedure.

1. Click [Start] [All Programs] [Accessories] and then select “Command Prompt”

2. Input the full path to setup.exe, together with the command for creating the installation log file (/r),
and then press the Enter key.

> [setup.exe (specified with the full path)] [/r]



3. The setup screen appears, and installation is performed in the usual way ("Install" on page 2-1). Specify the installation location.

4. Upon the completion of installation, the "setup.iss" log file will have been created in “C:\WINDOWS\”.

Installer settings

By executing the following commands from the application installer, the TM-S1000 driver can be installed automatically.

>[setup.exe (specified with the full path)] [/s /f1 “setup.iss (specified with the full path)”]

Ex: D:\TM-S1000>setup.exe /s /f1"D:\TM-S1000\setup.iss"

Setting non-display in (Add or Remove Programs)

When the EPSON TM-S1000 driver is installed by means of silent install, “TM-S1000 Driver Version x.xx” will appear in [Add or Remove Programs] ([Uninstall a Program] in Windows Vista or newer Windows versions.), in the same way as when it is installed normally. If you want to prevent it from appearing in [Add or Remove Programs], execute the installer with the following command. (Setting non-display can prevent the user from accidentally deleting the driver.)

>[setup.exe (specified with the full path)] [/s /f1”setup.iss(specified with the full path)”] [/z”/Invisible”]

Ex: D:\TM-S1000>setup.exe /s /f1”D:\TM-S1000\setup.iss” /z”/Invisible”

Uninstalling the driver forcibly

To uninstall the EPSON TM-S1000 Driver forcibly, add the following command to the installer.

>[setup.exe (specified with the full path)] [/z"uninstall"]

Ex: D:\TM-S1000>setup.exe /z"uninstall"

Comfirming the result of the Silent Install

After finishing the silent install, a log file (Setup.log) is created. You can confirm the result in the [ResponseResult] section in the "Setup.log" file. The log file is created in the following directory.

- ❏ Same directory as the installation log file
- ❏ Arbitrary directory
  By executing the following command from the application installer, a log file with a specified file name is created in a specified output directory.

  [setup.exe (specified with the full path)] [/s /f1"setup.iss(specified with the full path)"] [/f2"/output directory and file name"]

  Ex: D:\TM-S1000>setup.exe /s /f1"D:\TM-S1000\setup.iss" /f2"D:\TM-S1000\Result\Result.log"

The following table shows the result codes and the descriptions.

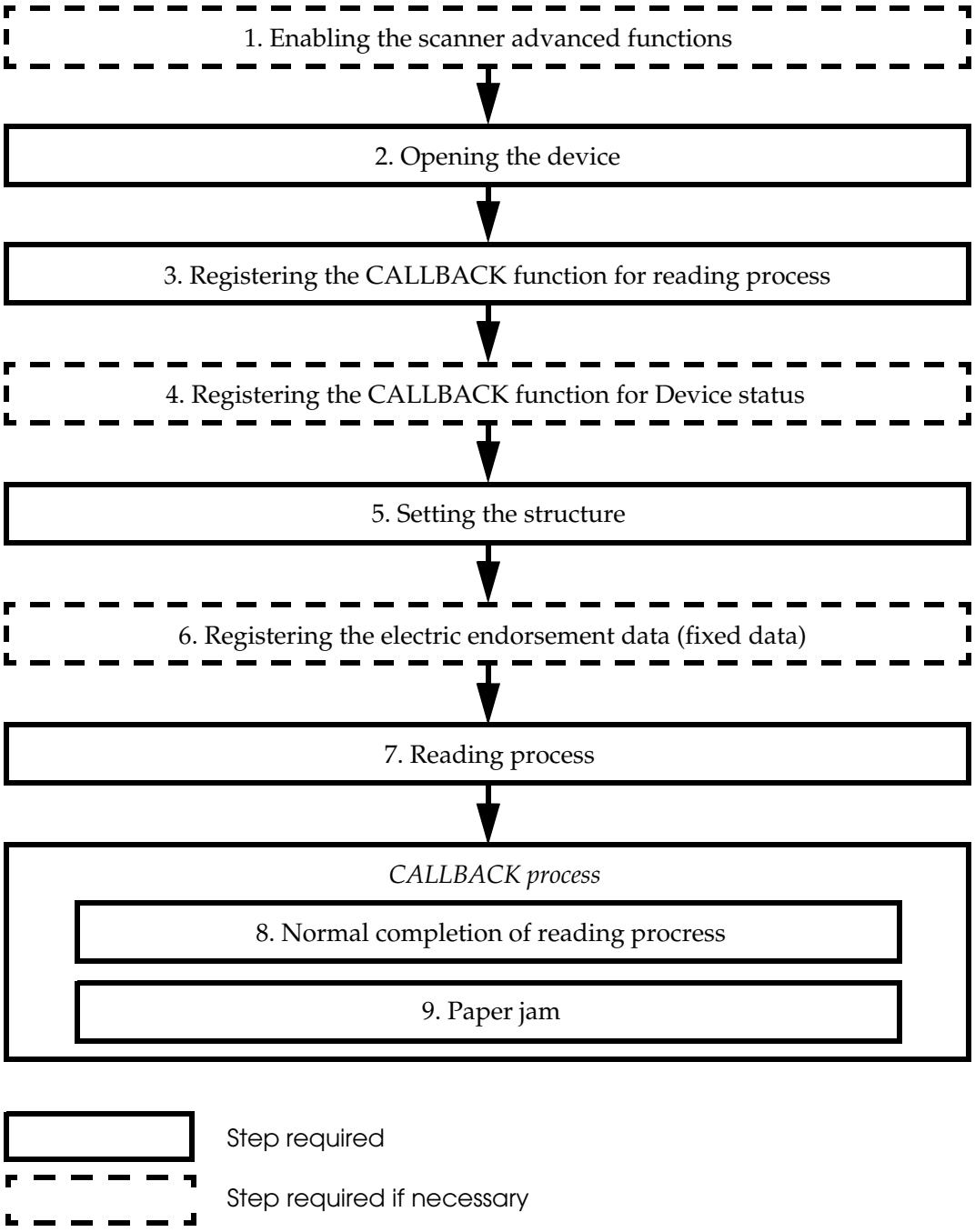
Result Code	Description
0	SUCCESS
-3	Specified data is not found in the installation log file (.iss).
-5	The installation log file (.iss) does not exist.
-8	The path for the installation log file (.iss) is not correct.

Chapter 3
Programming guide

This chapter describes how to program using the TM-S1000 API.

Application Processing Steps

This section describes basic processing steps of applications using the TM-S1000 API.



Each step operates as follows.

1. Enabling the scanner advanced functions

The scanner advanced functions allow you to edit scanned-in images as follows.
If none of the following operations is required, you do not need to use the advanced functions since scanned-in images are automatically processed. (the default automatic process crops out unnecessary part of the scanned-in image and rotates the image 90 degrees)

- ❏ Editing scanned-in images using customer's application
- ❏ Cropping specified area of the scanned-in image and saving the cropped image

See "How to Use the Scanner Advanced Functions" on page 3-72.

2. Opening the device

Search the connected TM-S1000 to obtain a handle. (See "Step 1-1. Process before use of the device" on page 3-20.)

3,4. Registering the CALLBACK function

There are two kinds of the CALLBACK function available for the TM-S1000 API.

- ❏ CALLBACK for reading process

  This CALLBACK function is called at the start/end of reading process, start/end of paper feeding process, start/end of data reception and a paper jam error.
  Set the CALLBACK function for reading process with BiSCNMICRSetStatusBackFunction. If the development environment is VB, set it with BiSCNMICRSetStatusBackWnd. (See "Step 1-1. Process before use of the device" on page 3-20.)

- ❏ CALLBACK for Device status

  This CALLBACK function is called when the sensor status of the TM-S1000 changes or a paper jam error occurs. (See "Step 7-1. Process before use of the device" on page 3-52 , "Device Status" on page 4-1.)

5. Setting the structure

Set values for each structure as shown below.

Structure	Settings
MF_BASE01	Buzzer setting
MF_SCAN	Scanning setting (resolution, size of character string to add, and so on)
MF_MICR	MICR setting (E13B/CMC7 and so on)
	OCR setting
MF_OCR_AB	OCR-A/B font setting
MF_PROCESS	Selecting the processing mode (High speed/Confirmation)
	Setting the operation when an error (double-feeding/insertion orientation/ noise/Bad Data) is detected.
	Setting the operation when a pocket near full is detected. To prevent paper jams in pockets, it is recommended to set MF_PROCESS.bNearFullSelect = MF_NEARFULL_NOT_PERMIT. (Default: MF_NEARFULL_PERMIT)
MF_IQA	IQA setting
MF_BARCODE	Barcode setting (Behavior at barcode decode/barcode decode error)

How to set initial values: See “Step 1-2. Reading process” on page 3-21.

How to set the buzzer: See “Step 6-1. Buzzer setting” on page 3-46.

How to set MICR: See “Step 5-1. Selecting the MICR font” on page 3-41.

How to set OCR-A/B font: See “Step 6-3. Obtaining/Displaying/Storing read data (CALLBACK process)” on page 3-48.

How to set MF_PROCESS: See “Step 4-1. Setting the Process When a Reading Error Occurs” on page 3-36.

How to set MF_IQA: See “Step 8-1. Setting the IQA process” on page 3-64.

How to set MF_BARCODE: See “Step 8-2. Setting the barcode decode” on page 3-66.

6. Registering the electric endorsement data (fixed data)

Register data (image/text) to paste on read front or back image data. (See “Step 3-2. Setting electric endorsement (fixed data)” on page 3-31.)
Registered electric endorsement data is automatically pasted on obtained image data.
If you want to specify pasting/not pasting in applications or change electric endorsement data, register electric endorsement data in "8. CALLBACK process (Normal completion of reading process)". (See “Step 3-4. Obtaining/Displaying read data (CALLBACK process)” on page 3-34).

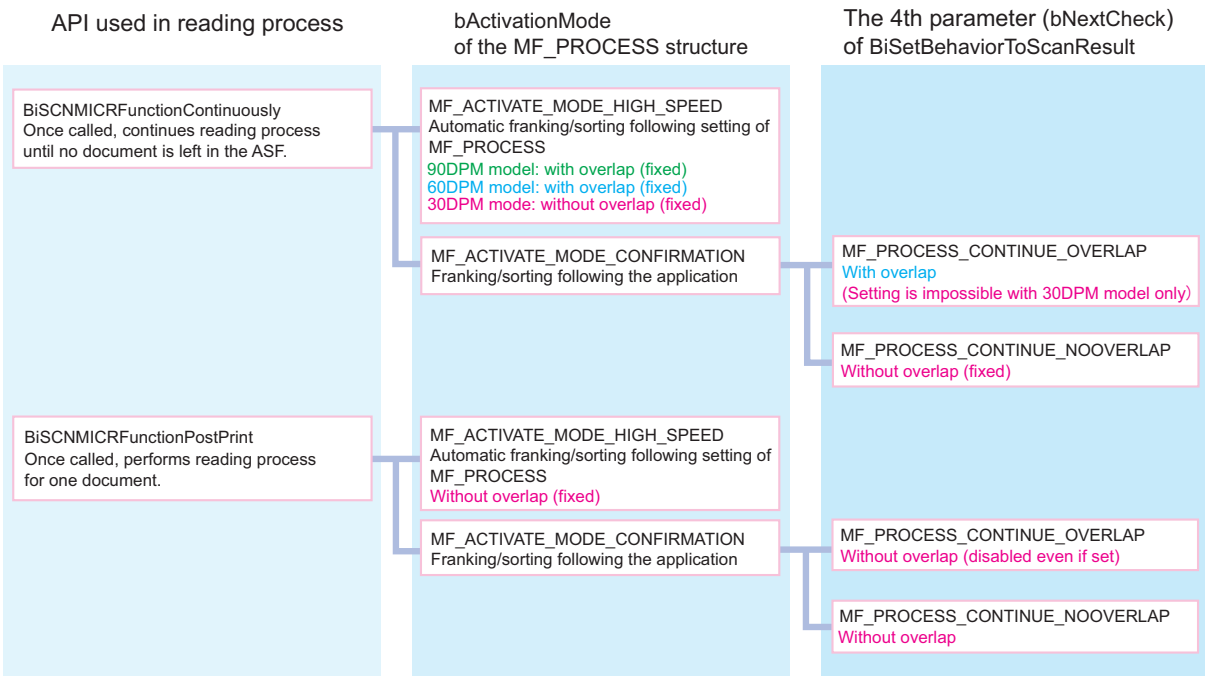
7. Reading process

Obtain a transaction number (See “Step 2-2. Obtaining/Displaying read data (CALLBACK process)” on page 3-26), and then use either one of the following APIs depending on the process mode to start the reading process. (See “Step 1-2. Reading process” on page 3-21, "Step 2-1. Reading process" on page 3-25, and "Step 3-3. Reading process" on page 3-33.)

Processing mode	API	Description
High-speed mode	BiSCNMICRFunctionContinuously	Automatically performs franking and sorting documents into the pockets without stopping feeding paper in the ASF.
Confirmation mode	BiSCNMICRFunctionContinuously BiSCNMICRFunctionPostPrint	Stops paper feeding for each paper, and performs franking and sorting document into the pockets following the application judgement.

When reading process has started, scanning both sides and MICR reading start at the same time.

Basic structure of API for reading process



<Note>
Overlap will not be realized even if BiSCNMICRFunctionPostPrint is executed repeatedly.

<Caution>
During processing with overlap, two documents may be in the paper path.
The transaction number tells you which document is jammed when a paper jam occurs.

8. CALLBACK process (Normal completion of reading process)

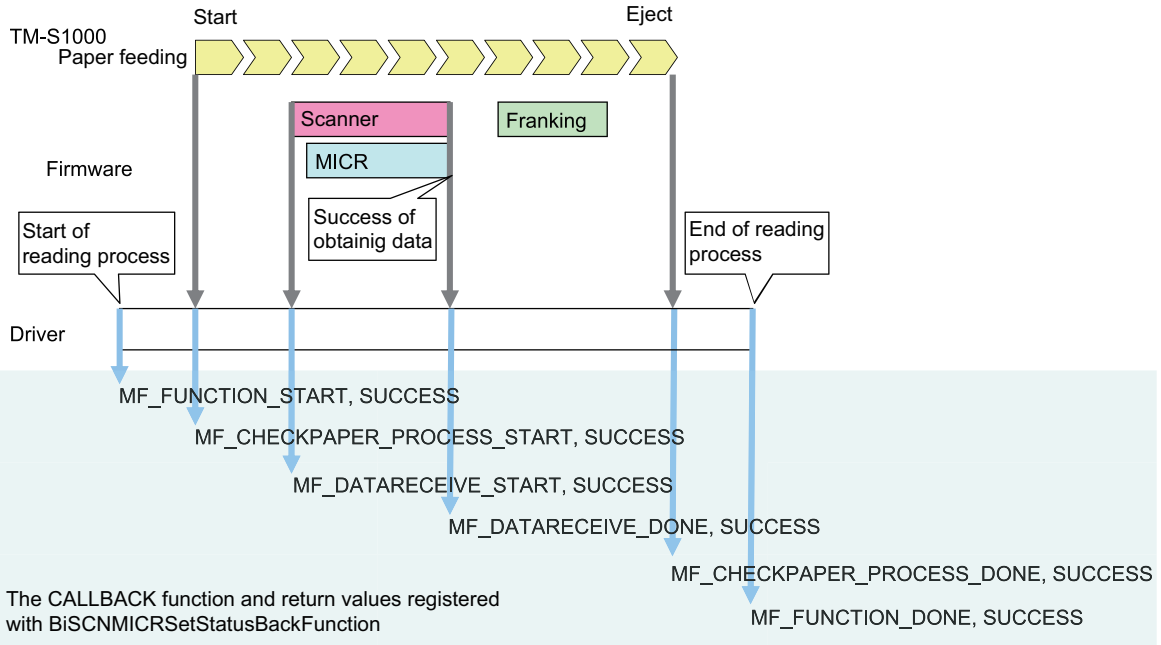
In this reading process, events (processing status) and return values return in the following order.

Event	Status	Return value
① Start of reading process	MF_FUNCTION_START	SUCCESS
② Start of paper feeding	MF_CHECHPAPER_PROCESS_START	SUCCESS
③ Start of data reception	MF_DATARECEIVE_START	SUCCESS
④ End of data reception	MF_DATARECEIVE_DONE	SUCCESS
⑤ End of paper feeding	MF_CHECKPAPER_PROCESS_DONE	SUCCESS
⑥ End of reading process	MF_FUNCTION_DONE	SUCCESS

After confirming the MF_DATARECEIVE_DONE status, perform the required process in the CALLBACK function following the steps below.

1. Obtaining MICR data. (See "Step 2-2. Obtaining/Displaying read data (CALLBACK process)" on page 3-26.)
2. Specifying image data format and obtaining front image data.(See "Step 2-2. Obtaining/Displaying read data (CALLBACK process)" on page 3-26.)
3. Pasting electric endorsement (different image/text for each reading) (See "Step 3-4. Obtaining/Displaying read data (CALLBACK process)" on page 3-34.)
4. Specifying the readable area and obtaining OCR-A/B data. (See "Step 6-3. Obtaining/Displaying/Storing read data (CALLBACK process)" on page 3-48.)
5. Specifying image data format and obtaining back image data. (See "Step 2-2. Obtaining/Displaying read data (CALLBACK process)" on page 3-26.)
6. In the confirmation mode, specifying the ejection pocket, franking/not franking, and overlapping/not overlapping (Use BiSetBehaviorToScnResult.). (See "Step 4-2. Obtaining/Displaying read data (CALLBACK process)" on page 3-39.)

The timing of return of events (processing status) and return values are shown below.



9. CALLBACK process (paper jam error occurred)

It is a paper jam error when paper is not detected at positions expected from feeding amount. The event (status) and return value differ depending on the timing of a paper jam error and whether read data is discarded or obtained.
This section describes process of the following 3 patterns. Troubleshooting for a paper jam error is also described.

- When a paper jam error occurs after end of data reception
- When a paper jam error occurs during obtaining data and option is set to discarded the read data.
- When a paper jam error occurs during obtaining data and option is set to keep the read data.
- How to recover from a paper jam error

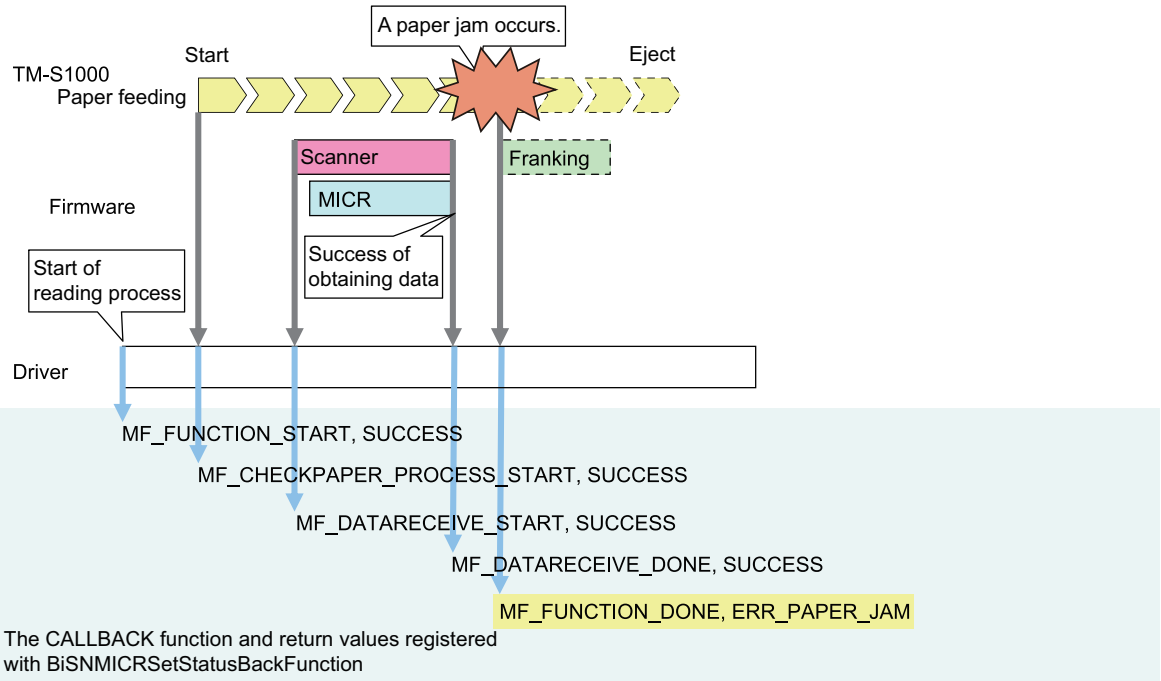
For error handling, see "Step 7-3. Error handling" on page 3-55.

When a paper jam error occurs after end of data reception

ERR_PAPER_JAM is returned as a return value of MF_FUNCTION_DONE.

End of data reception	MF_DATARECEIVE_DONE , SUCCESS
<<A paper jam error occurs.>>	
End of reading process	MF_FUNCTION_DONE , ERR_PAPER_JAM

The timing of return of events (processing status) and return values is shown below.



Note:
In this case, MF_CHECKPAPER_PROCESS_DONE is not returned.

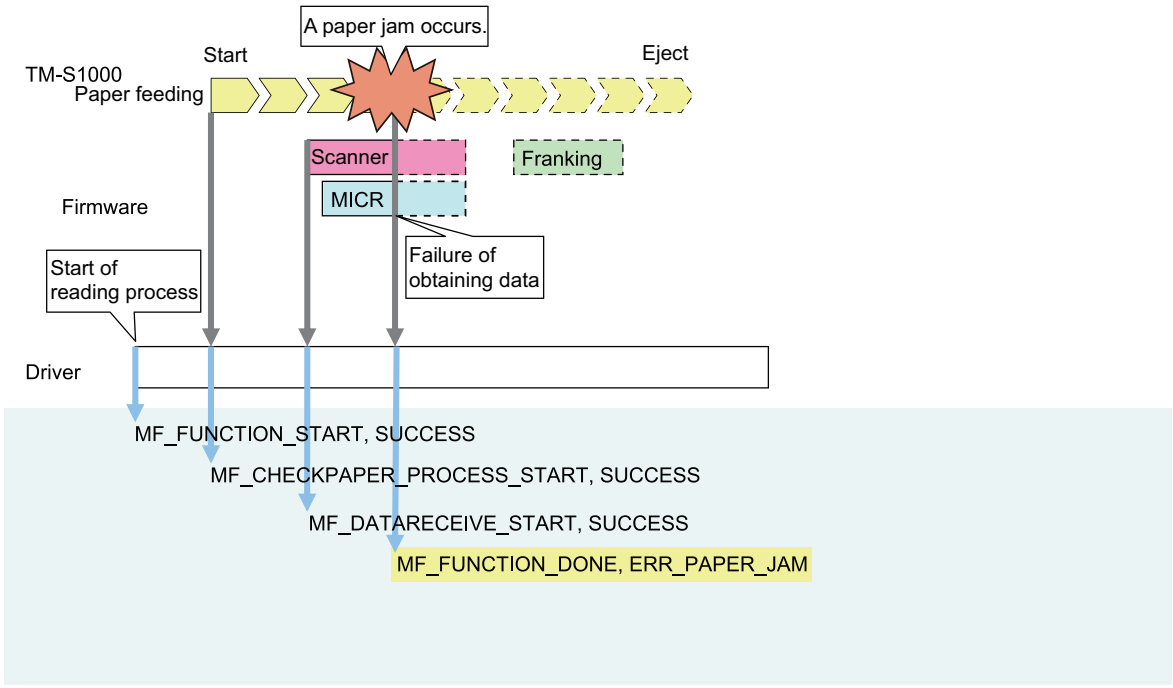
When a paper jam error occurs during obtaining data and option is set to discarded the read data

Note:
Specify MF_RESULT_NONE for bResultPartialData of the MF_PROCESS structure in advance to discard read data.

ERR_PAPER_JAM is returned as a return value of MF_FUNCTION_DONE.

Start of reading process	MF_DATARECEIVE_START , SUCCESS
<<A paper jam error occurs.>>	
End of reading process	MF_FUNCTION_DONE , ERR_PAPER_JAM

The timing of return of events (processing status) and return values is shown below.



Note:
In this case, MF_CHECKPAPER_PROCESS_DONE and MF_DATARECEIVE_DONE are not returned.

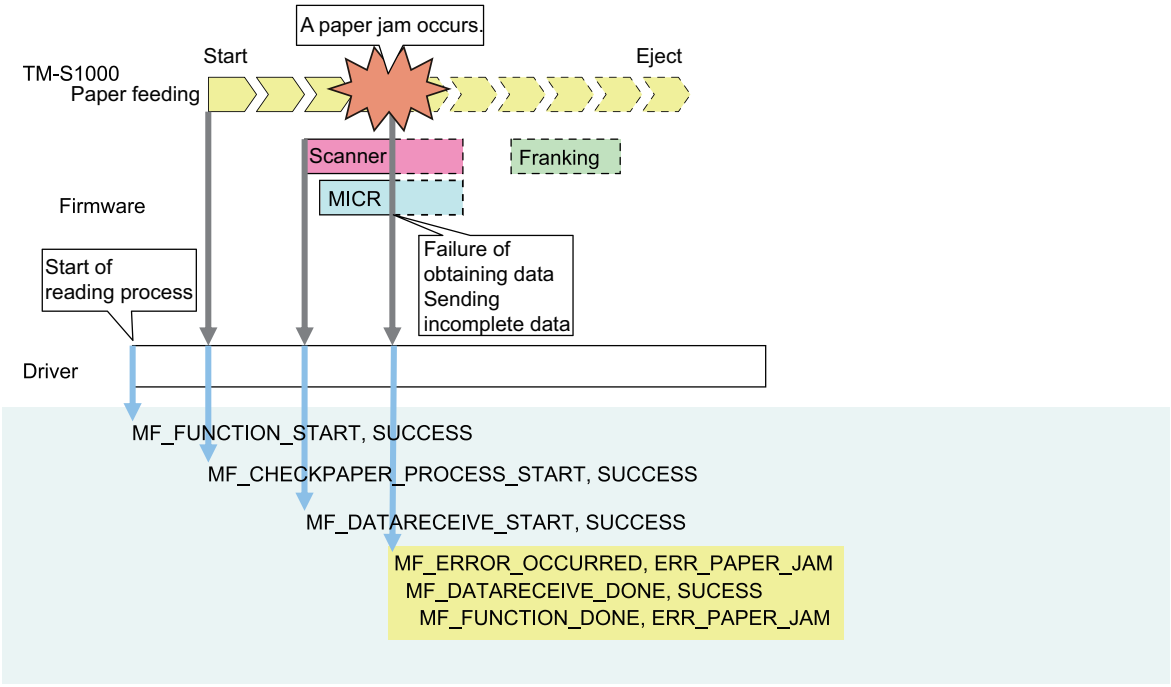
When a paper jam error occurs during obtaining data and option is set to keep the read data.

Note:
Specify MF_RESULT_PARTIAL for bResultPartialData of the MF_PROCESS structure in advance to obtain read data.

When a paper jam occurs, MF_ERROR_OCCURRED and ERR_PAPER_JAM are returned.

Start of reading process	MF_DATARECEIVE_START , SUCCESS
<<A paper jam error occurs.>>	
Error occurrence	MF_ERROR_OCCURRED , ERR_PAPER_JAM
End of data reception	MF_DATARECEIVE_DONE , SUCCESS
End of reading process	MF_FUNCTION_DONE , ERR_PAPER_JAM

The timing of return of events (processing status) and return values is shown below.



Note:
In this case, MF_CHECKPAPER_PROCESS_DONE is not returned.

How to recover from a paper jam error

Open the covers of the TM-S1000 and remove the paper in the paper path. Then, call BiCancelError from the application to recover from a paper jam error.
Confirm the sensor status with the CALLBACK function of device status (See "3,4. Registering the CALLBACK function" on page 3-2.) or BiGetStatus to display a message prompting users to remove jammed paper.

MF_PROCESS structure

By setting values for members of the MF_PROCESS structure, setting the eject pocket or franking/not franking is possible. Pay attention to the following items.

Detectable item	Detection	Judgement condition
① Double feed	Firmware	The paper thickness is more than specified with bPaperType or has changed.
② Insertion orientation error	Firmware	The firmware analyzes the magnetic waveform and judges a upside down and wrong side (front/back) of documents. (wrong side: only for E13B)
③ Noise	Firmware	Noises are detected in the magnetic waveform.
④ No data	API	The magnetic waveform was not detected.
⑤ Bad data	API	Unanalyzable characters were detected more than the number specified with bBaddataCount.

- ❏ Enabling ① the double feeding detection is also necessary to detect ② the insertion orientation error.
- ❏ When the detection of ④ No data and ⑤ Bad data is enabled, setting the ejection pocket or franking operation reduces the processing speed.
- ❏ The priority order of the detection is; ① Double feed is the highest and ⑤ Bad data is the lowest. When more than one detection is enabled, the operation setting whose priority is the highest has the priority.

  Example: When the following two settings are set.

  - If insertion orientation error is detected: Ejection into the main pocket
  - If No data is detected: Ejection into the sub pocket

  When the both are detected at the same time, the former setting has the priority and documents are ejected into the main pocket even if No data is detected.
- ❏ If a reading error occurs when the ejection pocket is set to “no ejection,” the TM-S1000 stops feeding the document and goes to the recoverable error status (See the TM-S1000 Technical Reference Guide.) and the error LED flashes. In this case, open the covers of the TM-S1000 and remove the documents. And then call BiCancelError from the application to recover from the recoverable error.
- ❏ If none of magnetic waveform is detected from the paper, both ② the insertion orientation error and ④ No data error occur.

MF_IQA structure

MF_IQA structure is a structure that performs IQA (Image Quality Assurance) analysis of scanned data. IQA is a standard of the FSTC (Financial Services Technology Consortium). Pay attention to the following description.

- ❏ The priority order of error detection
  MF_IQA detects IQA error after the error detection by MF_PROCESS structure is finished. When MF_PROCESS structure detects an error, the operation set for MF_PROCESS has priority.

- ❏ Confirmation of the result of IQA validation
  Setting for MF_IQA is executed when image data is scanned. By calling BiGetIQAResult at that time, IQA validation result (MF_IQA_RESULT structure) can be confirmed.

- ❏ MF_IQA structure setting

Content	Member	Description
IQA function	bErrorSelect	Enables/Disables IQA function.
Operation when IQA error is detected	bErrorEject	Sets the ejection pockets for a document.
	bStamp	Sets the franking process.
	bCancel	Sets whether the reading process of a document is continued/canceled.
Image quality of IQA validation setting	bImageFormat	Sets an image format.
	bColorDepth	Sets the gradation.
	bThreshold	Sets the density threshold.
	bColor	Sets color.
	bExOption	Sets the variety of density adjustment.
	sResolution	Sets the resolution.
Detected items	bUndersize	Enables/Disables UndersizeImage validation.
	bOversize	Enables/Disables OversizeImage validation.
	bMincompressed	Enables/Disables MinCompressedImageSize validation.
	bMaxcompressed	Enables/Disables MaxCompressedImageSize validation.
	bFront_rear	Enables/Disables FrontRearImageMismatch validation.
	bToolight	Enables/Disables ImageTooLight validation.
	bToodark	Enables/Disables ImageTooDark validation.
	bStreaks	Enables/Disables HorizontalStreaksPresent validation.
	bNoise	Enables/Disables ExcessiveSpotNoise validation.
	bFocus	Enables/Disables ImageOutOfFocus validation.
	bCorners	Enables/Disables FoldedTornDocCorners validation.
	bEdges	Enables/Disables FoldedTornDocEdges validation.
	bFraming	Enables/Disables DocFramingError validation.
	bSkew	Enables/Disables ExcessiveDocSkew validation.
	bCarbon	Enables/Disables CarbonStripDetection validation.
	bPiggyback	Enables/Disables Piggyback validation.

MF_BARCODE structure

MF_BARCODE structure is a structure that decodes a barcode data from the scanned image data. Pay attention to the following description when using it.

Note:
Decoding a barcode data decelerates the scanning speed. For the details, refer to "API used for each processing mode and its setting" on page 3-14.

❏ Barcode decoding setting
Before scanning, configure the following settings for this structure.

- Specifying the barcode symbols to decode
  By specifying the barcodes to decode, the deceleration of the scanning speed can be reduced.

- Specifying the decoding directions
  Specify the direction referring to the following table. The decoded result will be returned in the decoded order.

Decoding direction (bDirection)	Description
MF_BARCODE_DIRECTION_ALL	When decoding horizontally lined barcodes, they are decoded from left to right. When decoding vertically lined barcodes, they are decoded from top to bottom. If both horizontally lined barcodes and vertically lined barcodes exist, they are decoded from left to right.
MF_BARCODE_DIRECTION_LEFTRIGHT	Barcodes are decoded from left to right. If barcodes are lined vertically, they are decoded from top to bottom.
MF_BARCODE_DIRECTION_TOPBOTTOM	Barcodes are decoded from top to bottom. If barcodes are lined horizontally, they are decoded from right to left.
MF_BARCODE_DIRECTION_RIGHTLEFT	Barcodes are decoded from right to left. If barcodes are lined vertically, they are decoded from bottom to top.
MF_BARCODE_DIRECTION_BOTTOMTOP	Barcodes are decoded from bottom to top. If barcodes are lined horizontally, they are decoded from left to right.

- Configuring the error remedy process when decoding fails
  Configures enabling/disabling the decode error detection, selection of ejection pocket, franking, and canceling the scanning operation.

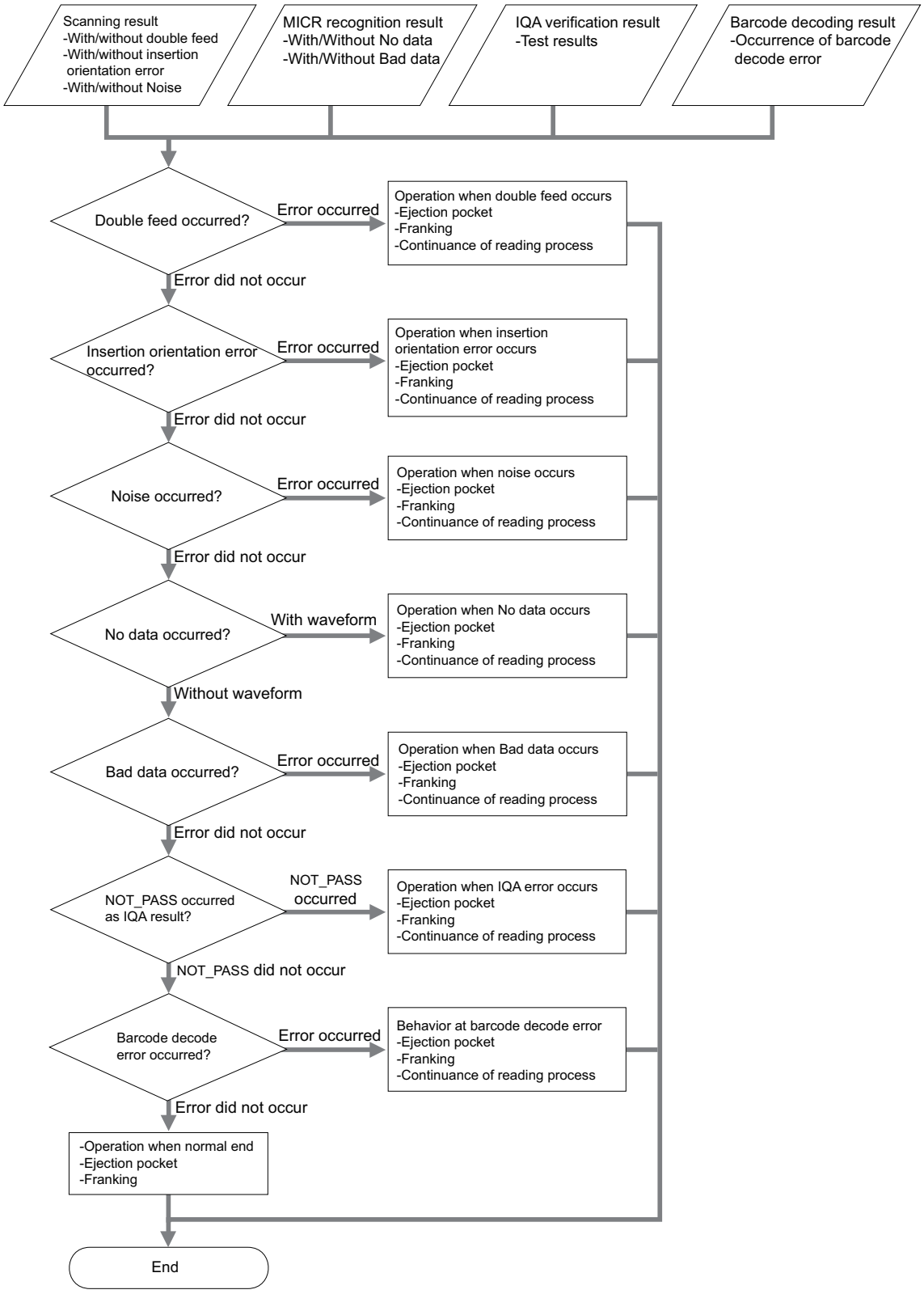
❏ Decoding barcodes and acquiring barcode data
Use the following APIs when decoding barcodes and acquiring barcode data.

- BiGetBarcodeData: API for decoding when scanning
- BiDecodeBarcode: API for decoding the scanned image file

❏ Error remedy process when decoding fails
The barcode decode error is processed according to the setting of this structure.
This error is detected the last. That is, if a bad data error or the like occurs in advance when the decoding is successfully complete, this decoding is regarded as an error. For the details, refer to "Error detections and operation priorities" on page 3-12.

Error detections and operation priorities

Error detections (Double feed/Insertion orientation error/Noise/No data/Bad data/IQA/Barcode decode error) and operation priorities when an error occurs are shown below.



Operation when an error occurs in each mode

<table>
<tr><th rowspan="2">Process when an error occurs</th><th colspan="3">Operation setting when an error occurs</th></tr>
<tr><th>High-speed mode</th><th>Confirmation mode</th><th>Waterfall mode</th></tr>
<tr><td>Sorting to ejection pockets</td><td colspan="2" rowspan="3">Setting of MF_PROCESS / MF_IQA/ MF_BARCmODE structure</td><td>Setting of BiSetWaterfallMode*</td></tr>
<tr><td>Franking</td><td rowspan="2">Setting ofMF_PROCESS/ MF_IQA/ MF_BARCODE structure</td></tr>
<tr><td>Reading cancel</td></tr>
</table>

* If the ejection setting when an error occurs of MF_PROCESS / MF_IQA/ MF_BARCODE structure is MF_EJECT_NOEJECT, the printer operates following the setting of MF_PROCESS / MF_IQA/ MF_BARCODE structure.

API used for each processing mode and its setting

API to use and API setting differs depending on the TM-S1000 models and processing modes. Refer to the following description to create applications for your purposes.

Processing modes for 30 dpm models

<table>
<tr><th colspan="3">Processing mode</th><th colspan="2">High-speed mode</th><th>Waterfall mode</th><th>Confirmation mode</th></tr>
<tr><td colspan="3">Function</td><td colspan="3">Automatically performs franking and sorting documents into the pockets without stopping feeding paper in the ASF.</td><td>Stops paper feeding for each paper, and performs franking and sorting document into the pockets following the application judgement.</td></tr>
<tr><td colspan="3">Overlap*1</td><td colspan="3">Does not perform.</td><td>Does not perform.</td></tr>
<tr><td colspan="3">Detectable items</td><td colspan="3">Double feed/insertion orientation error/noise /No Data/Bad Data</td><td>*3</td></tr>
<tr><td rowspan="4">Processing speed *2</td><td rowspan="2">IQA</td><td>Disabled</td><td colspan="3">30 dpm</td><td rowspan="2">Maximum 28 DPM *3</td></tr>
<tr><td>Enabled</td><td colspan="3">❒ When document sorting/franking is performed: 28 dpm
❒ When document sorting/franking is not performed: 30 dpm</td></tr>
<tr><td rowspan="2">Barcode</td><td>Disabled</td><td colspan="3">30 dpm</td><td rowspan="2">Maximum 28 DPM *3</td></tr>
<tr><td>Enabled</td><td colspan="3">❒ When document sorting/franking is performed: 28 dpm
❒ When document sorting/franking is not performed: 30 dpm</td></tr>
<tr><td colspan="3" rowspan="2">API</td><td colspan="3" rowspan="2">BiSCNMICRFunctionContinuously</td><td>BiSCNMICRFunctionContinuously</td></tr>
<tr><td>BiSCNMICRFunctionPostPrint</td></tr>
<tr><td colspan="3">bActivationMode, a member of MF_PROCESS structure*4</td><td colspan="3">MF_ACTIVATE_MODE_HIGH_SPEED</td><td>MF_ACTIVATE_MODE_CONFIRMATION</td></tr>
<tr><td colspan="3">The forth parameter of BiSetBehaviorToScnResult (bNextCheck)</td><td colspan="3">Does not use the API described at the left.</td><td>MF_PROCESS_CONTINUE_NOOVERLAP</td></tr>
<tr><td colspan="3" rowspan="2">Waterfall (BiSetWaterfallMode)</td><td colspan="2" rowspan="2">WATERFALL_MODE_DISABLE</td><td>WATERFALL_MODE_STANDARD</td><td rowspan="2">*5</td></tr>
<tr><td>WATERFALL_MODE_INHERIT_POCKET</td></tr>
</table>

*1 Overlap
Performs: While feeding a document, starts feeding the next document.
Does not perform: After ejecting a document, starts feeding the next document.

*2 Enabling the detection of No Data/Bad Data described above reduces the processing speed. For details, see "MF_PROCESS structure" on page 3-9.

*3 Depends on the application.

*4 MF_PROCESS structure
One of the structures the TM-S1000 API has. Setting values for its members can change the processing mode or enable automatic franking process or sorting documents into the pockets when an error (double feed/insertion orientation error/noise/No Data/Bad Data) is detected.

*5 Makes settings for Waterfall mode when using the Waterfall function in Confirmation mode.

Processing modes for 60 dpm models

<table>
<tr><th colspan="3">Processing mode</th><th colspan="2">High-speed mode</th><th>Waterfall mode</th><th>Confirmation mode (without overlap)</th><th>Confirmation mode (with overlap)</th></tr>
<tr><td colspan="3">Function</td><td colspan="3">Automatically performs franking and sorting documents into the pockets without stopping feeding paper in the ASF.</td><td>Stops paper feeding for each paper, and performs franking and sorting document into the pockets following the application judgement.</td><td>Stops paper feeding for each paper, and performs franking and sorting document into the pockets following the application judgement.</td></tr>
<tr><td colspan="3">Overlap*1</td><td colspan="3">Performs.</td><td>Does not perform.</td><td>Performs.</td></tr>
<tr><td colspan="3">Detectable items</td><td colspan="3">Double feed/insertion orientation error/noise/No Data/Bad Data</td><td colspan="2">*3</td></tr>
<tr><td rowspan="4">Processing speed *2</td><td rowspan="2">IQA</td><td>Disabled</td><td colspan="3">60 dpm</td><td rowspan="2">28 DPM *3</td><td>40 DPM *3</td></tr>
<tr><td>Enabled</td><td colspan="3">❒ When document sorting/franking is performed: 32 dpm
❒ When document sorting/franking is not performed:60 dpm</td><td>32 DPM *3</td></tr>
<tr><td rowspan="2">Barcode</td><td>Disabled</td><td colspan="3">60 dpm</td><td rowspan="2">28 DPM *3</td><td>40 DPM *3</td></tr>
<tr><td>Enabled</td><td colspan="3">❒ When document sorting/franking is performed: 32 dpm
❒ When document sorting/franking is not performed:60 dpm</td><td>32 DPM *3</td></tr>
<tr><td colspan="3" rowspan="2">API</td><td colspan="3" rowspan="2">BiSCNMICRFunctionContinuously</td><td>BiSCNMICRFunction Continuously</td><td rowspan="2">BiSCNMICRFunction Continuously</td></tr>
<tr><td>BiSCNMICRFunctionPostPrint</td></tr>
<tr><td colspan="3">bActivationMode, a member of MF_PROCESS structure*4</td><td colspan="3">MF_ACTIVATE_MODE_HIGH_SPEED</td><td>MF_ACTIVATE_MODE_CONFIRMATION</td><td>MF_ACTIVATE_MODE_CONFIRMATION</td></tr>
<tr><td colspan="3">The forth parameter of BiSetBehaviorToScnResult (bNextCheck)</td><td colspan="3">Does not use the API described in the Processing mode column.</td><td>MF_PROCESS_CONTINUE_NOOVERLAP</td><td>MF_PROCESS_CONTINUE_OVERLAP</td></tr>
<tr><td colspan="3" rowspan="2">Waterfall (BiSetWaterfallMode)</td><td colspan="2" rowspan="2">WATERFALL_MODE_DISABLE</td><td>WATERFALL_MODE_STANDARD</td><td colspan="2" rowspan="2">*5</td></tr>
<tr><td>WATERFALL_MODE_INHERIT_POCKET</td></tr>
</table>

*1 Overlap
Performs: While feeding a document, starts feeding the next document.
Does not perform: After ejecting a document, starts feeding the next document.

*2 Enabling the detection of No Data/Bad Data described above reduces the processing speed. For details, see "MF_PROCESS structure" on page 3-9.

*3 Depends on the application.

*4 MF_PROCESS structure
One of the structures the TM-S1000 API has. Setting values for its members can change the processing mode or enable automatic franking process or sorting documents into the pockets when an error (double feed/insertion orientation error/noise/No Data/Bad Data) is detected.

*5 Makes settings for Waterfall mode when using the Waterfall function in Confirmation mode.

Processing modes for 90 dpm models

<table>
<tr><th colspan="3">Processing mode</th><th colspan="2">High-speed mode</th><th>Waterfall mode</th><th>Confirmation mode (without overlap)</th><th>Confirmation mode (with overlap)</th></tr>
<tr><td colspan="3">Function</td><td colspan="2">Automatically performs franking and sorting documents into the pockets without stopping feeding paper in the ASF.</td><td></td><td>Stops paper feeding for each paper, and performs franking and sorting document into the pockets following the application judgement.</td><td>Stops paper feeding for each paper, and performs franking and sorting document into the pockets following the application judgement.</td></tr>
<tr><td colspan="3">Overlap*1</td><td colspan="2">Performs.</td><td></td><td>Does not perform.</td><td>Performs.</td></tr>
<tr><td colspan="3">Detectable items</td><td colspan="2">Double feed/insertion orientation error/noise/No Data/Bad Data</td><td></td><td colspan="2">*3</td></tr>
<tr><td rowspan="4">Processing speed *2</td><td rowspan="2">IQA</td><td>Disabled</td><td colspan="2">✻ Personal Check: 90 DPM
✻ Business Check: 75 DPM</td><td></td><td rowspan="2">28 DPM *3</td><td>40 DPM *3</td></tr>
<tr><td>Enabled</td><td colspan="2">❒ When document sorting/franking is performed:32 dpm
❒ When document sorting/franking is not performed:
✻ Personal Check: 90 DPM
✻ Business Check: 75 DPM</td><td></td><td>32 DPM *3</td></tr>
<tr><td rowspan="2">Barcode</td><td>Disabled</td><td colspan="2">✻ Personal Check: 90 DPM
✻ Business Check: 75 DPM</td><td></td><td rowspan="2">28 DPM *3</td><td>40 DPM *3</td></tr>
<tr><td>Enabled</td><td colspan="2">❒ When document sorting/franking is performed:32 dpm
❒ When document sorting/franking is not performed:
✻ Personal Check: 90 DPM
✻ Business Check: 75 DPM</td><td></td><td>32 DPM *3</td></tr>
<tr><td colspan="3" rowspan="2">API</td><td colspan="2" rowspan="2">BiSCNMICRFunctionContinuously</td><td></td><td>BiSCNMICRFunctionContinuously</td><td rowspan="2">BiSCNMICRFunctionContinuously</td></tr>
<tr><td></td><td>BiSCNMICRFunctionPostPrint</td></tr>
<tr><td colspan="3">bActivationMode, a member of MF_PROCESS structure*4</td><td colspan="2">MF_ACTIVATE_MODE_HIGH_SPEED</td><td></td><td>MF_ACTIVATE_MODE_CONFIRMATION</td><td>MF_ACTIVATE_MODE_CONFIRMATION</td></tr>
<tr><td colspan="3">The forth parameter of BiSetBehaviorToScnResult (bNextCheck)</td><td colspan="2">Does not use the API described in the Processing mode column.</td><td></td><td>MF_PROCESS_CONTINUE_NOOVERLAP</td><td>MF_PROCESS_CONTINUE_OVERLAP</td></tr>
<tr><td colspan="3" rowspan="2">Waterfall (BiSetWaterfallMode)</td><td rowspan="2">WATERFALL_MODE_DISABLE</td><td>WATERFALL_MODE_STANDARD</td><td></td><td colspan="2">*5</td></tr>
<tr><td>WATERFALL_MODE_INHERIT_POCKET</td><td></td><td colspan="2"></td></tr>
</table>

*1 Overlap
Performs: While feeding a document, starts feeding the next document.
Does not perform: After ejecting a document, starts feeding the next document.

*2 Enabling the detection of No Data/Bad Data described above reduces the processing speed. For details, see "MF_PROCESS structure" on page 3-9.

*3 Depends on the application.

*4 MF_PROCESS structure
One of the structures the TM-S1000 API has. Setting values for its members can change the processing mode or enable automatic franking process or sorting documents into the pockets when an error (double feed/insertion orientation error/noise/No Data/Bad Data) is detected.

*5 Makes settings for Waterfall mode when using the Waterfall function in Confirmation mode.

Sample Programs

Programming with the TM-S1000 API is described using 8 functional level sample programs.

Step 1: Opening/Closing the device
Describes how to execute the device opening/closing process and reading process.

Step 2: Displaying read data
In addition to Step 1, describes how to display read image and MICR data on the application screen.

Step 3: Continuous reading/Electric endorsement
In addition to Step 2, describes how to set the processing methods (continuous reading/one-by-one reading) and electric endorsement.

Step 4: Process setting when a reading error occurs
In addition to Step 3, describes process setting such as how to sort documents into the two pockets automatically when a reading error occurs or how to process differently depending on a read result in an application.

Step 5: Selecting the MICR font and setting the image quality
In addition to Step 4, describes selecting a MICR font and setting the image quality.

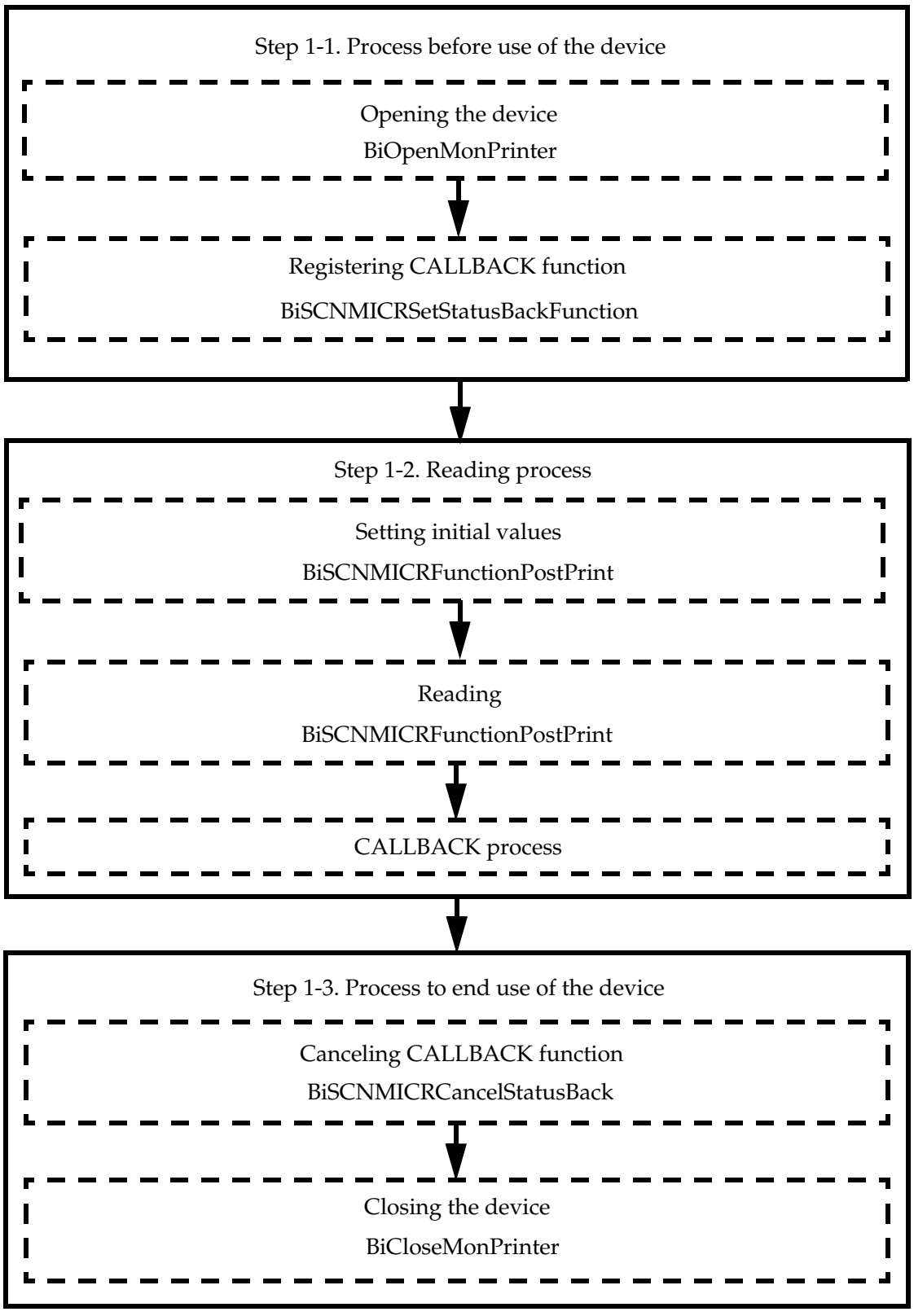
Step 6: Reading OCR-A/B font and buzzer setting
In addition to Step 5, describes how to read an OCR-A/B font with the OCR function and buzzer setting.

Step 7: Confirming the Device status and error handling
In addition to Step 6, describes how to confirm the device status, how to handle errors (pocket near-full/paper jam error), and how to process MICR cleaning.

Step 8: Decoding a barcode, confirming the IQA and Waterfall process
In addition to Step 7, describes how to decode a barcode, how to confirm the IQA and how to process Waterfall.

Step 1 Opening/Closing the Device

Process opening and closing the device. Also execute the reading process. The read data will be displayed in Step 2.



Step 1-1. Process before use of the device

Perform the following processes before using the device:

- Opening the device
- Registering the CALLBACK function

Opening device

Specify a device type for the first parameter and specify a device name for the second parameter of BiOpenMonPrinter, and then the TM-S1000 will be searched and a handle will be obtained.

m_iHandle = BiOpenMonPrinter (TYPE_PRINTER, “TM-S1000U”)

Note:

A handle is returned to m_iHandle in the sample programs. Hereafter, the handle is used for the other functions. If the searched TM-S1000 is used by the other application, ERR_Access is returned to m_iHandle. Specify the device type, “TYPE_PRINTER” for compatibility with the TM-J9000 API although a printer is not installed on the TM-S1000.

Programing code

APIUsage.cpp CAPIUsage::CAPIUsage()

```
m_iHandle = BiOpenMonPrinter(TYPE_PRINTER, “TM-S1000U”);
if(m_iHandle < SUCCESS){
        ::MessageBox(NULL,_T("Unable to connect to printer\n\nEnsure Driver installed
                correctly\nand Printer is powered on"),GetResultString(m_iHandle),MB_OK);
return;
}
```

Registering the CALLBACK function

Forsuch incidents as starting/ending document processing or data reception, the TM-S1000 API causes an event (For details on events, see "8. CALLBACK process (Normal completion of reading process)" on page 3-5.) Register the CALLBACK function that is called when the event occurs. Specify the handle obtained above for the first parameter and CALLBACK function name for the second parameter of BiSCNMICRSetStatusBackFunction to register the CALLBACK function.

nErr= BiSCNMICRSetStatusBackFunction (m_iHandle, cbScanStatus)

Note:

In the sample programs, a CALLBACK function named cbScanStatus is specified. The status when an event occurs can be obtained in this CALLBACK function.

Programming code

APIUsage.cpp

```
int CALLBACK cbScanStatus(DWORD dwTransactionNumber, WORD wMainStatus, WORD wSubStatus,
                                                                        LPSTR pPortName)
{
        CTMS1000SampleDlg* pDlg = (CTMS1000SampleDlg*)(theApp.m_pMainWnd);
        if(pDlg != NULL){
                pDlg->m_api.ScanStatus(dwTransactionNumber, wMainStatus, wSubStatus, pPortName);
        }
        return 0;
}
```

APIUsage.cpp CAPIUsage::CAPIUsage()

```
CheckResponse(BiSCNMICRSetStatusBackFunction(m_iHandle, cbScanStatus));
```

Step 1-2. Reading process

Perform the reading process (image and MICR reading) of the TM-S1000 and confirm the processing status with the CALLBACK function. How to display the read data is described in Step 2.

Initial value setting

Specify initial values for structures using BiSCNMICRFunctionPostPrint, one of the functions that execute the reading process.

1. Specify the memory address of each structure for the second parameter and each parameter as shown below for the third parameter of BiSCNMICRFunctionPostPrint to call the initial value for each structure. (For detailed information on initial values, see "BiSCNMICRFunctionContinuously" on page 4-58.) How to change the initial values is described in Step 4.

   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, lpvStruct, MF_GET_xxxx_DEFAULT)

Structure	Description	Parameter to call
MF_BASE01	Structure of the TM-S1000 basic operation	MF_GET_BASE_DEFAULT 0x0030
MF_SCAN	Structure of the scanning function	MF_GET_SCAN_FRONT_DEFAULT 0x0032 MF_GET_SCAN_BACK_DEFAULT 0x0033
MF_MICR	Structure of the MICR function	MF_GET_MICR_DEFAULT 0x0031
MF_PRINT01	Structure of the electric endorsement function	MF_GET_PRINT_DEFAULT 0x0034
MF_PROCESS	Structure of the optional functions	MF_GET_PROCESS_DEFAULT 0x0035

2. Call BiSCNMICRFunctionPostPrint as follows to set values for each structure.

   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, lpvStruct, MF_SET_xxxx_PARAM)

   Note:
   Set for the MF_BASE01 at first.
   The initial value of the lpString in MF_PRINT01 is NULL. If the initial value is not changed, ERR_PARAM will return to nErr. Set a value other than NULL.

Programing code

APIUsage.cpp CAPIUsage::Configure()

```
// Base
m_tBase01.iSize = sizeof(MF_BASE01);
m_tBase01.iVersion = MF_BASE_VERSION01;
CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tBase01, MF_GET_BASE_DEFAULT));
CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tBase01, MF_SET_BASE_PARAM));
```

```
        // Scan front
        m_tScanFront.iSize = sizeof(MF_SCAN);
        m_tScanFront.iVersion = MF_SCAN_VERSION;
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tScanFront, MF_GET_SCAN_DEFAULT));
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tScanFront,
MF_SET_SCAN_FRONT_PARAM));

        // Scan back
        m_tScanBack.iSize = sizeof(MF_SCAN);
        m_tScanBack.iVersion = MF_SCAN_VERSION;
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tScanBack, MF_GET_SCAN_DEFAULT));
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tScanBack,
MF_SET_SCAN_BACK_PARAM));

        // Micr
        m_tMicr.iSize = sizeof(MF_MICR);
        m_tMicr.iVersion = MF_MICR_VERSION;
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tMicr, MF_GET_MICR_DEFAULT));
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tMicr, MF_SET_MICR_PARAM));

        // Print
        m_tPrint.iSize = sizeof(MF_PRINT01);
        m_tPrint.iVersion = MF_PRINT_VERSION01;
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tPrint, MF_GET_PRINT_DEFAULT));
        // If the pString values of the m_tPrint structure are all NULL, an error occurs.
        m_tPrint.lpString[0] = "";              // Specify a null character
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tPrint, MF_SET_PRINT_PARAM));

        // Process
        m_tProcess.iSize = sizeof(MF_PROCESS);
        m_tProcess.iVersion = MF_PROCESS_VERSION;
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tProcess,
MF_GET_PROCESS_DEFAULT));
        CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tProcess, MF_SET_PROCESS_PARAM));
```

Reading process

Specify a handle for the first parameter, NULL for the second parameter, and MF_EXEC for the third parameter of BiSCNMICRFunctionPostPrint to start the reading process.

nErr= BiSCNMICRFunctionPostPrint (m_iHandle , NULL , MF_EXEC)

Programming code

APIUsage.cpp CAPIUsage::Scan()

```
CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, NULL, MF_EXEC));
```

CALLBACK process

After reading documents has started, the registered CALLBACK function will be called every time an event occurs when starting reading process, starting document processing, starting data reception, ending reception of reading data, ending document processing, and ending reading process.
If MF_FUNCTION_DONE status (an event when ending reading process) is confirmed in CALLBACK function, reading process has finished.

Note:
You need to confirm the return value of MF_FUNCTION_DONE to judge whether the reading process was successful or an error occurred.

Programming code

CAPIUsage.cpp CAPIUsage::ScanStatus()

```
void CAPIUsage::ScanStatus(DWORD dwTransactionNumber, WORD wMainStatus, WORD wSubStatus,
                                                                    LPSTR pPortName)
{
    if(wMainStatus == MF_FUNCTION_DONE){
        ::SetEvent(m_hScanEvent);
    }
```

Step 1-3. Process to end use of the device

Perform the following processes to end using the device:

- Cancelling the CALLBACK function
- Closing the device

Cancelling the CALLBACK function

Specify a handle for the parameter of BiSCNMICRCancelStatusBack to cancel the registered CALLBACK function.

```
nErr = BiSCNMICRCancelStatusBack (m_iHandle)
```

Programming code

CAPIUsage.cpp CAPIUsage::CancelScan()

```
BiSCNMICRCancelStatusBack(m_iHandle);
```

Closing the device

Specify a handle for the parameter of BiCloseMonPrinter to close the device.

```
nErr = BiCloseMonPrinter (m_iHandle)
```

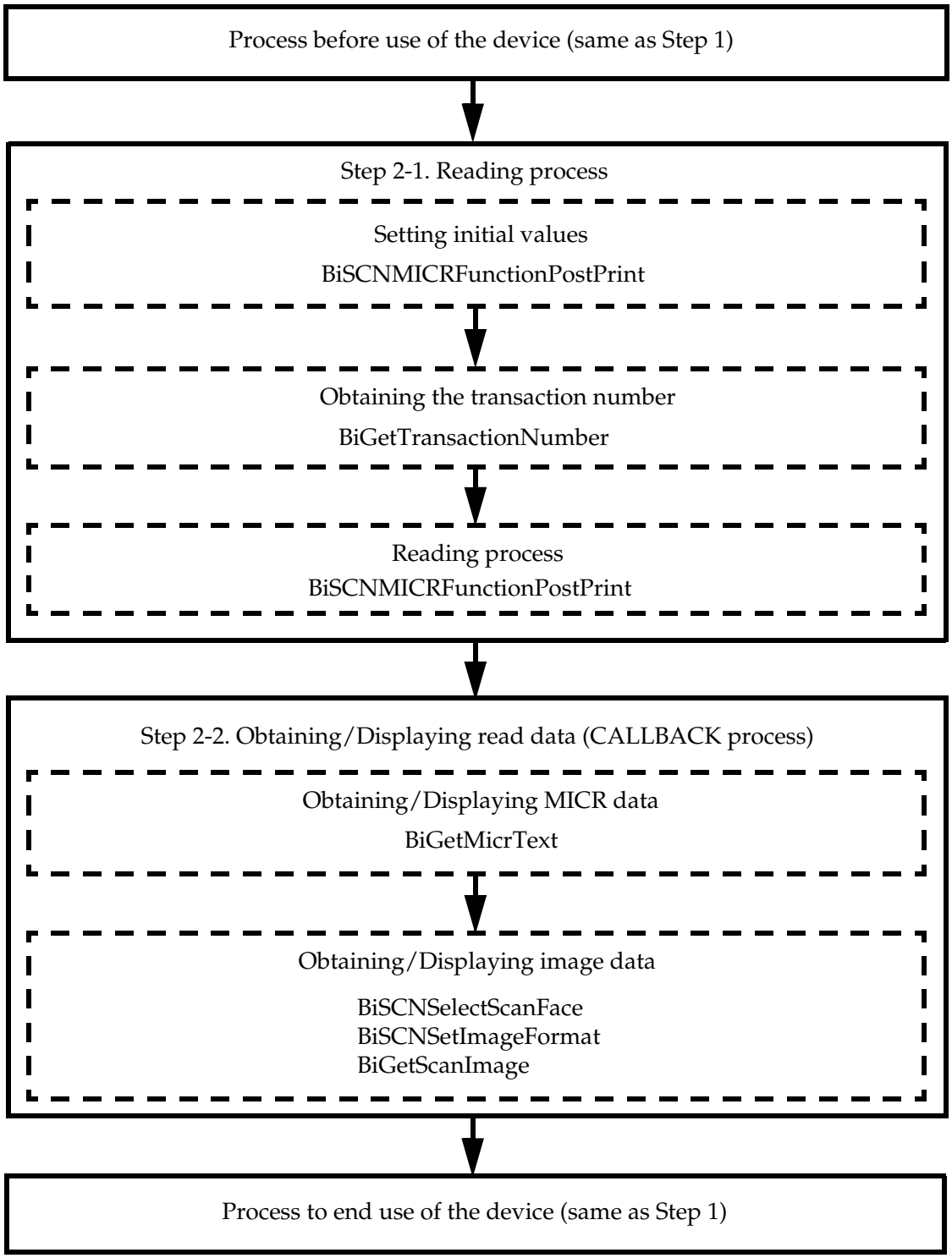
Programming code

CAPIUsage.cpp CAPIUsage::CancelScan()

```
BiCloseMonPrinter(m_iHandle);
```

Step 2 Displaying the Read Data

In addition to Step 1, display read images and MICR data on the application screen.



Step 2-1. Reading process

Follow the steps below to perform the reading process.

- Initial value setting (same as Step 1)
- Obtaining the transaction number
- Reading process (image + MICR)

Obtaining the transaction number

Specify the memory address where a transaction number is stored for the second parameter of BiGetTransactionNumber to obtain the transaction number that is used for reading process. Every time the BiGetTransactionNumber is called, the transaction number is incremented.

nErr= BiGetTransactionNumber (m_iHandle, lpdwTransactionNumber)

Programming code

CAPIUsage.cpp CAPIUsage::GetTransactonNumber()

```
DWORD dwTransactionNumber = 0;
BiGetTransactionNumber(m_iHandle, &dwTransactionNumber);
```

As the result, 1 returns to *dwTransactionNumber* as the transaction number.

Reading image/MICR

Specify NULL for the second parameter and MF_EXEC for the third parameter of BiSCNMICRFunctionPostPrint to execute image and MICR reading.

nErr= BiSCNMICRFunctionPostPrint (m_iHandle, NULL , MF_EXEC)

Programming code

APIUsage.cpp CAPIUsage::Scan()

```
CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, NULL, MF_EXEC));
```

Note:
BiSCNMICRFunctionPostPrint performs reading image and MICR at the same time.
In Step 2, readable MICR font is E13B and readable image is 2 values, black and white. These settings can be changed in Step 5.
The TM-S1000 stops scanning operation when 10 scanned-in images are stored in the driver. The TM-S1000 resumes the scanning operation when the number of stored images becomes two or fewer. Therefore, the application should process the scanned-in images quickly.

Step 2-2. Obtaining/Displaying read data (CALLBACK process)

Obtain and display read data with the CALLBACK function.

- Confirming data reception end
- Obtaining/Displaying MICR data
- Obtaining/Displaying image data

Confirming data reception end

Confirm the MF_DATARECEIVE_DONE status with the CALLBACK function to confirm data reception end.

Obtaining/Displaying MICR data

Follow the steps below to obtain MICR data.

1. Specify MF_MICR_USE_MICR (use the magnetic head for reading MICR characters) for bMicOcrSelect of the MF_MICR structure.

   MF_MICR.bMicOcrSelect = MF_MICR_USE_MICR

   Note:
   The magnetic head (USE_MICR) or OCR (USE_OCR) is selectable for MICR reading. However, usually select the magnetic head (USE_MICR) for a higher recognition rate.

2. Specify the transaction number for the second parameter and the memory address of the MF_MICR structure for the third parameter of BiGetMicrText to obtain MICR data.

   nErr = BiGetMicrText (m_iHandle , dwTransactionNumber , ptMicr)

3. MICR data returns to szMicrStr of the MF_MICR structure.

4. MICR data is obtained in the CALLBACK function and displayed on the application.

Programming code

APIUsage.cpp CAPIUsage::ScanStatus()

```
m_tMicr.bMicOcrSelect = MF_MICR_USE_MICR;
iResult = BiGetMicrText(m_iHandle, dwTransactionNumber, &m_tMicr);
TCHAR pMicr[1024];
_tcscpy(pMicr, m_tMicr.szMicrStr);
```

Obtaining/Displaying image data

Each image data of the front and back side is individually obtained and displayed. Follow the steps below.

1. Specify the side to obtain for the second parameter of BiSCNSelectScanFace.
   nErr= BiSCNSelectScanFace (m_iHandle , MF_SCAN_FACE_*xxxx*)

MF_SCAN_FACE_xxxx	Description
MF_SCAN_FACE_FRONT (0)	Selects the front side (default)
MF_SCAN_FACE_BACK (1)	Selects the back side

2. Specify the image data format for the second parameter of BiSCNSetImageFormat.

   nErr= BiSCNSetImageFormat (m_iHandle, EPS_BI_SCN_xxxx)

EPS_BI_SCN_xxxx	Description
EPS_BI_SCN_TIFF(1)	TIFF format CCITT (Group 4) compressed data (default)
EPS_BI_SCN_BITMAP(3)	Bitmap format uncompressed data

3. Specify the transaction number for the second parameter and the memory address of the MF_SCAN structure for the third parameter to obtain image data.

   nErr = BiGetScanImage (m_iHandle, dwTransactionNumber, ptScan)

4. Image data returns to lpbScanData and the size of the image data returns to dwScanSize of the MF_SCAN structure.

5. Image data is obtained and displayed by the application.

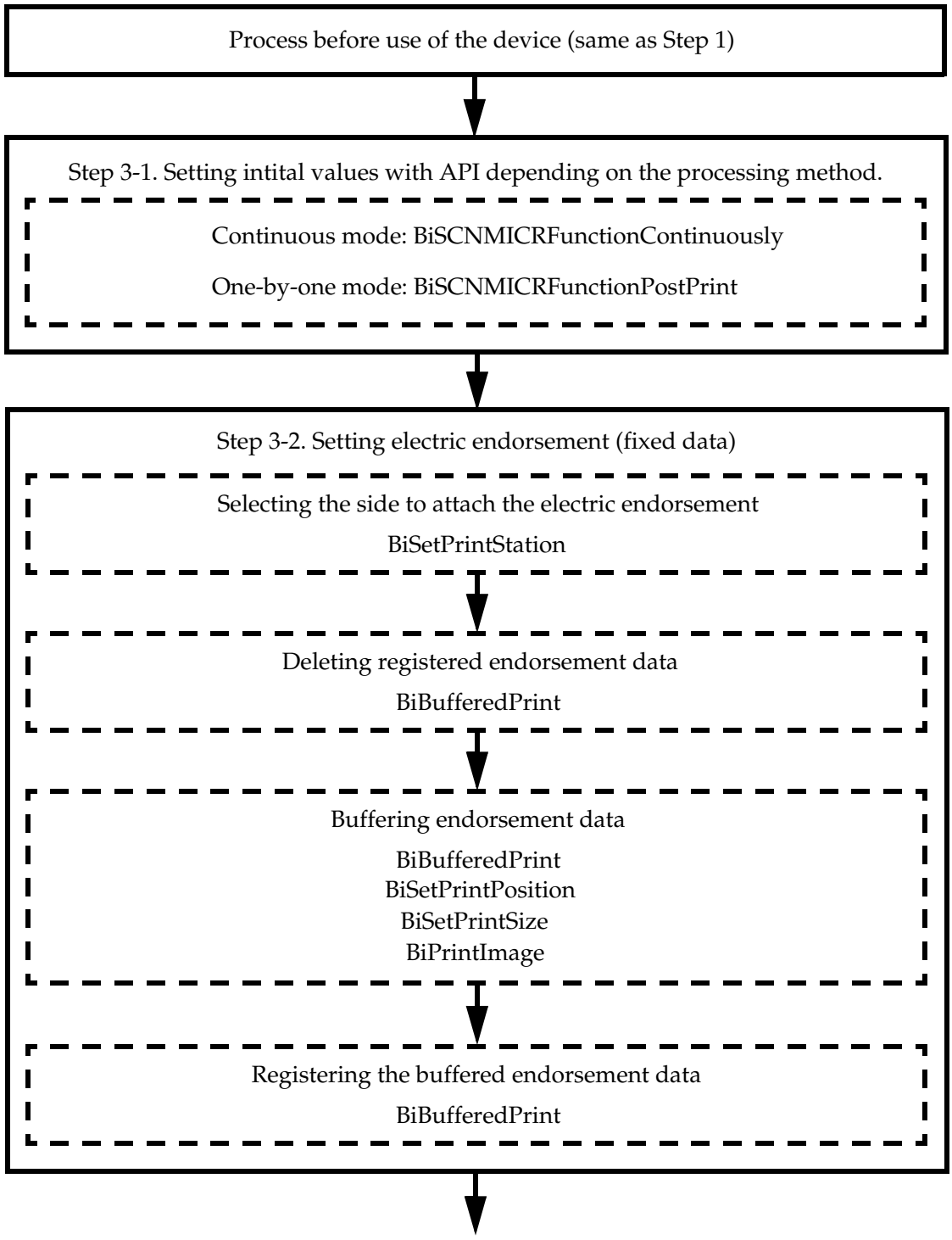
Programming code

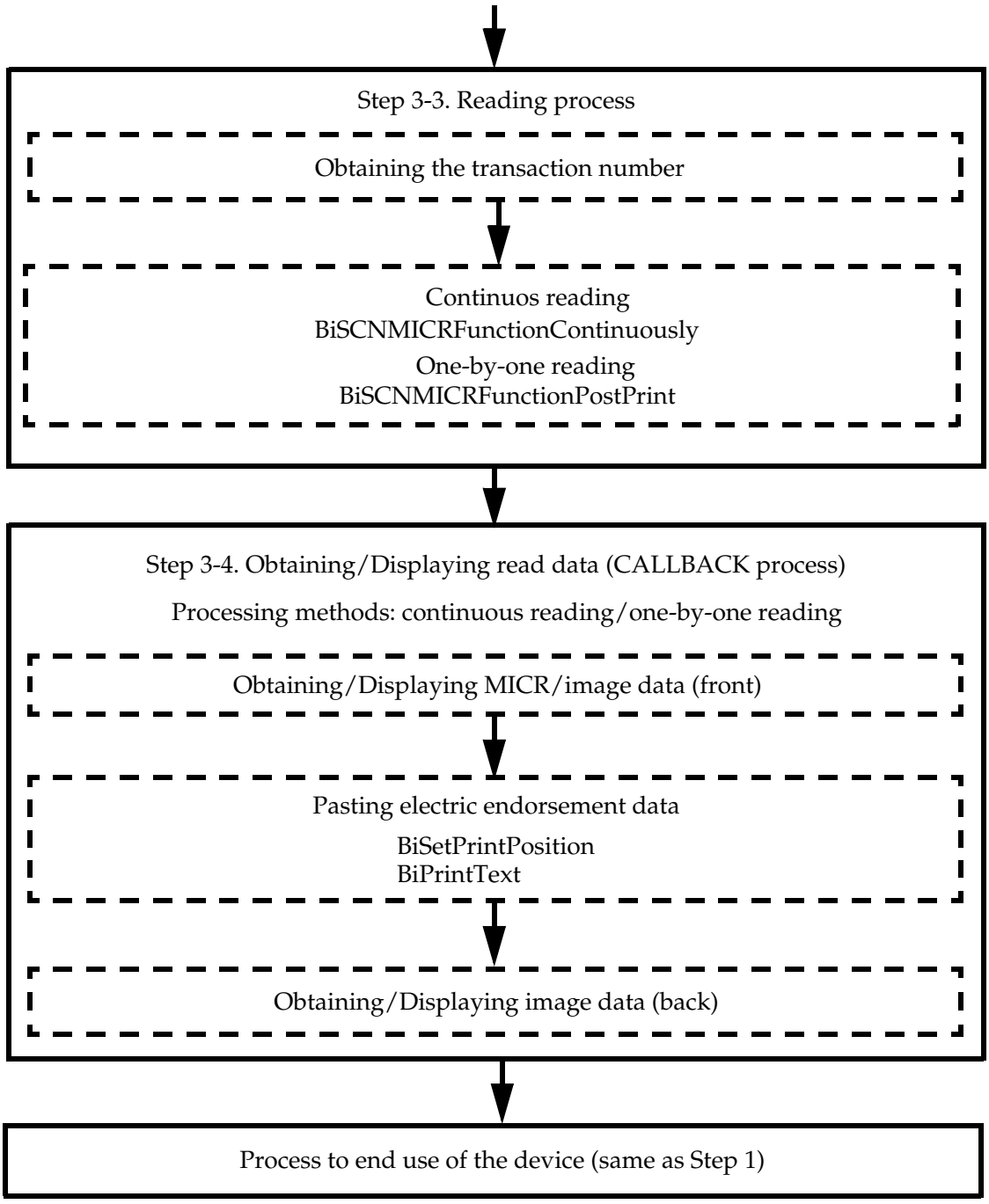
APIUsage.cpp CAPIUsage::ScanStatus()

```
iResult = BiSCNSelectScanFace(m_iHandle, MF_SCAN_FACE_FRONT);
if(iResult == SUCCESS){
        iResult = BiSCNSetImageFormat(m_iHandle, EPS_BI_SCN_BITMAP);
        if(iResult == SUCCESS){
                iResult = BiGetScanImage(m_iHandle, dwTransactionNumber, &m_tScanFront);
                if(iResult == SUCCESS && m_tScanFront.lpbScanData != NULL){
                        pDlg->SetImage(m_tScanFront.lpbScanData, m_tScanFront.dwScanSize,
                                                                                TRUE);
                        GlobalFree(m_tScanFront.lpbScanData);
                        m_tScanFront.lpbScanData = NULL;
                }
        }
}
```

Step 3 Continuous Reading/Electric Endorsement

In addition to Step 2, set the processing method (continuous reading/one-by-one reading) and electric endorsement.




Step 3-3. Reading process
Obtaining the transaction number
Continuos reading
BiSCNMICRFunctionContinuously
One-by-one reading
BiSCNMICRFunctionPostPrint
Step 3-4. Obtaining/Displaying read data (CALLBACK process)
Processing methods: continuous reading/one-by-one reading
Obtaining/Displaying MICR/image data (front)
Pasting electric endorsement data
BiSetPrintPosition
BiPrintText
Obtaining/Displaying image data (back)
Process to end use of the device (same as Step 1)

Step 3-1. Setting the processing method

The TM-S1000 has the following two processing methods. Initialize structures with API depending on the processing method.

Processing method	API to use	Description
Continuous reading	BiSCNMICRFunctionContinuously	Once called, continues reading until no paper is left in the ASF. Another API can cancel the reading.
One-by-one reading	BiSCNMICRFunctionPostPrint	Once called, reads only one sheet of paper.

Continuous reading

With BiSCNMICRFunctionContinuously, call and set initial values of the structures the same way as in Step 1-2.

1. Call initial values of structures

   nErr= BiSCNMICRFunctionContinuously (m_iHandle, lpvStruct,
   MF_GET_xxxx_DEFAULT)

2. Set the initial values

   nErr= BiSCNMICRFunctionContinuously (m_iHandle, lpvStruct MF_SET_xxxx_PARAM)

   Note:
   Process data stored in the driver within 500ms after the data is acquired to prevent the processing speed from slowing down.

One-by-one reading

With BiSCNMICRFunctionPostPrint, call and set initial values of structures in the same way as Step 1-2.

1. Call initial values of structures
   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, *lpvStruct*, MF_GET_xxxx_DEFAULT)

2. Set the initial values
   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, *lpvStruct*, MF_SET_xxxx_PARAM)

Step 3-2. Setting electric endorsement (fixed data)

Fixed data for electric endorsement that can be registered. (If you need to use different endorsement data depending on reading result, See "Step 3-4. Obtaining/Displaying read data (CALLBACK process)" on page 3-34.)

- Select the side to attach the electric endorsement
- Delete registered endorsement data
- Buffer endorsement data
- Register buffered endorsement data

Selecting the side to attach the electric endorsement

Specify the side to attach the electric endorsement for the second parameter of BiSetPrintStation.

nErr= BiSetPrintStation(*m_iHandle , wStation*)

wStation	Description
MF_ST_E_ENDORSEMENT	Attaches to the back side.
MF_ST_E_ENDORSEMENT_BACK	Attaches to the back side.
MF_ST_E_ENDORSEMENT_FRONT	Attaches to the back side.

Programming code

CAPIUsage.cpp CAPIUsage::Configure()

```
CheckResponse(BiSetPrintStation(m_iHandle, MF_ST_E_ENDORSEMENT));
```

Deleting registered endorsement data

Specify MF_PRT_CLEAR for the second parameter of BiBufferedPrint to delete electric endorsement data that is already registered.

nErr = BiBufferedPrint (m_iHandle, MF_PRT_CLEAR)

Programming code

CAPIUsage.cpp CAPIUsage::Configure()

```
CheckResponse(BiBufferedPrint(m_iHandle, MF_PRT_CLEAR));
```

Buffering endorsement data

1. Specify MF_PRT_BUFFERING for the second parameter of BiBufferedPrint.

   nErr = BiBufferedPrint (m_iHandle, MF_PRT_BUFFERING)

Programming code

CAPIUsage.cpp	CAPIUsage::ConfigureMultiple()
CAPIUsage.cpp	CAPIUsage::ConfigureSingle()

```
CheckResponse(BiBufferedPrint(m_iHandle, MF_PRT_BUFFERING));
```

2. Specify the horizontal direction for the second parameter and vertical direction for the third parameter of BiSetPrintPosition for the pasting starting position.

   nErr = BiSetPrintPosition (m_iHandle, wHorizontal, wVertical)

3. Specify the horizontal direction for the second parameter and vertical direction for the third parameter of BiSetPrintSiz for the pasting size.

   nErr = BiSetPrintSize (m_iHandle, wWidth, wHeight)

4. Specify image data for the second parameter of BiPrintImage to buffer it.

   nErr = BiPrintImage (m_iHandle, pFileName)

Note:
Specify the image file using the full path.
Use BiPrintText to specify text. (See Step 3-4.)
Specify the rotation direction of the electric endorsement by calling BiPrintImage BiSetEndorseDirection before calling BiSetEndorseDirection.

Programming code

APIUsage.cpp	CAPIUsage::ConfigureEndorseImage()

```
CheckResponse(BiSetPrintPosition(m_iHandle, 100, 30));
CheckResponse(BiSetPrintSize(m_iHandle, 30, 30));
CheckResponse(BiPrintImage(m_iHandle, "Image.jpg"));
```

Registering buffered endorsement data

By specifying MF_PRT_EXEC for BiBufferedPrint for the second parameter, specified endorsement data is registered and attached to the image data specified for BiSetPrintStation each time reading is performed.

nErr = BiBufferedPrint (m_iHandle, MF_PRT_EXEC)

Programming code

CAPIUsage.cpp	CAPIUsage::ConfigureMultiple()
CAPIUsage.cpp	CAPIUsage::ConfigureSingle()

```
CheckResponse(BiBufferedPrint(m_iHandle, MF_PRT_EXEC));
```

Step 3-3. Reading process

For continuous reading

Specify NULL for the second parameter and MF_EXEC for the third parameter of BiSCNMICRFunctionContinuously to start the reading process.

nErr= BiSCNMICRFunctionContinuously (m_iHandle, NULL, MF_EXEC)

Note:

Obtaining/displaying read data is performed by the CALLBACK process.

Programming code

APIUsage.cpp CAPIUsage::ScanMultiple()

```
CheckResponse(BiSCNMICRFunctionContinuously(m_iHandle, NULL, MF_EXEC));
```

For one-by-one reading

Specify NULL for the second parameter and MF_EXEC for the third parameter of BiSCNMICRFunctionPostPrint to start the reading process.

nErr= BiSCNMICRFunctionPostPrint (m_iHandle, NULL, MF_EXEC)

Note:

Obtaining/displaying read data is performed by the CALLBACK process.

Programming code

APIUsage.cpp CAPIUsage::ScanSingle()

```
CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, NULL, MF_EXEC));
```

Step 3-4. Obtaining/Displaying read data (CALLBACK process)

As in Step 2, confirm the MF_DATARECEIVE_DONE with the CALLBACK function, and then obtain MICR data and front image data. By following the steps below before obtaining image data, the endorsement data can be attached on the side of the image data specified for BiSetPrintStation each time reading is performed.

Pasting endorsement data

1. Specify the horizontal direction for the second parameter and the vertical direction for the third parameter of BiSetPrintPosition for the pasting starting position.

   nErr = BiSetPrintPosition (m_iHandle, wHorizontal, wVertical)

2. Specify the print data for the second parameter and the memory address (character decoration information) of the DECORATE structure for the third parameter of BiPrintText to paste text endorsement data.

   nErr = BiPrintText (m_iHandle, szText, tDecorate)

Note:
Specify the rotation direction of the electric endorsement by calling BiSetEndorseDirection before calling BiPrintText.

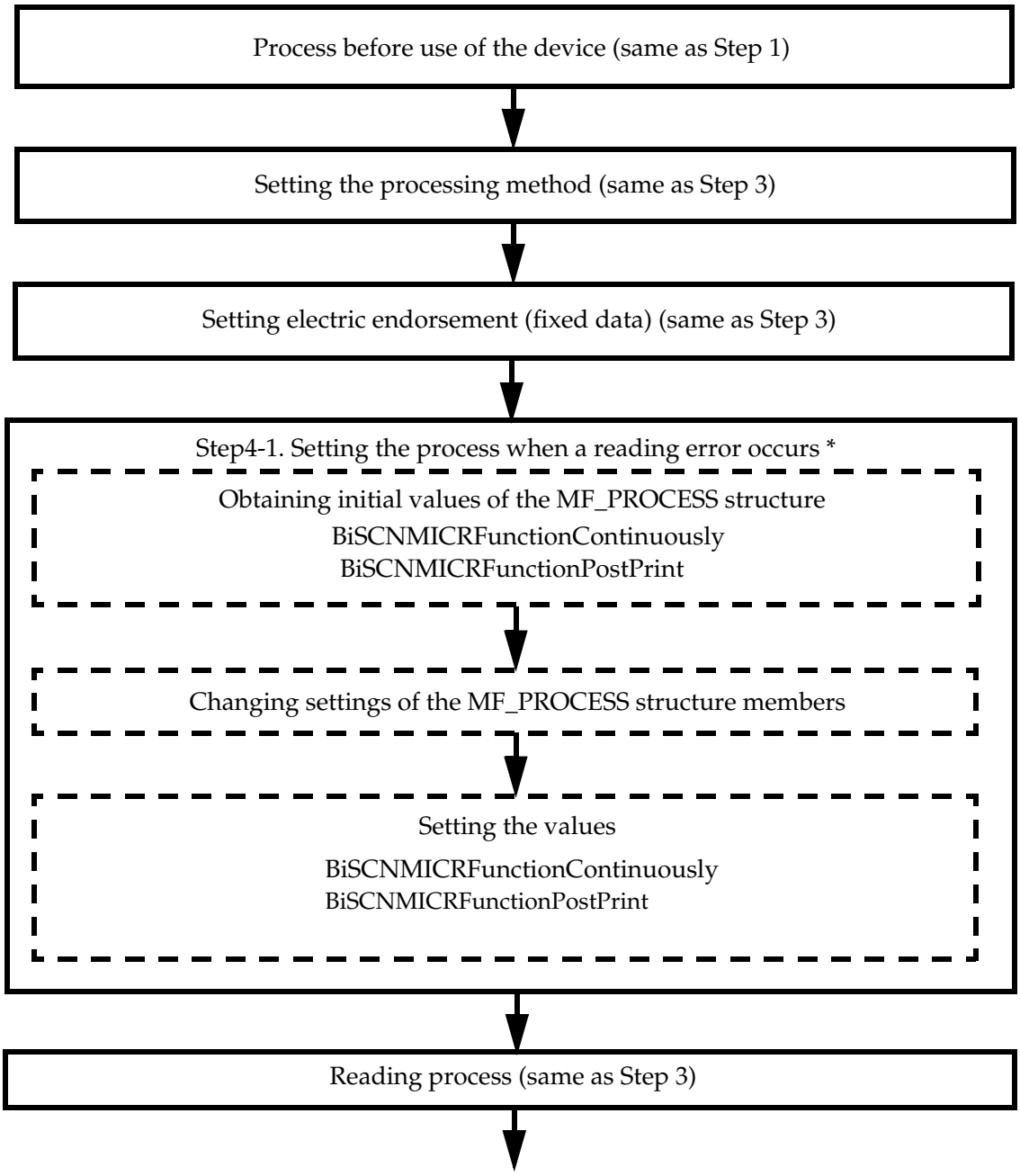
Programming code

APIUsage.cpp CAPIUsage::ConfigureEndorseText()

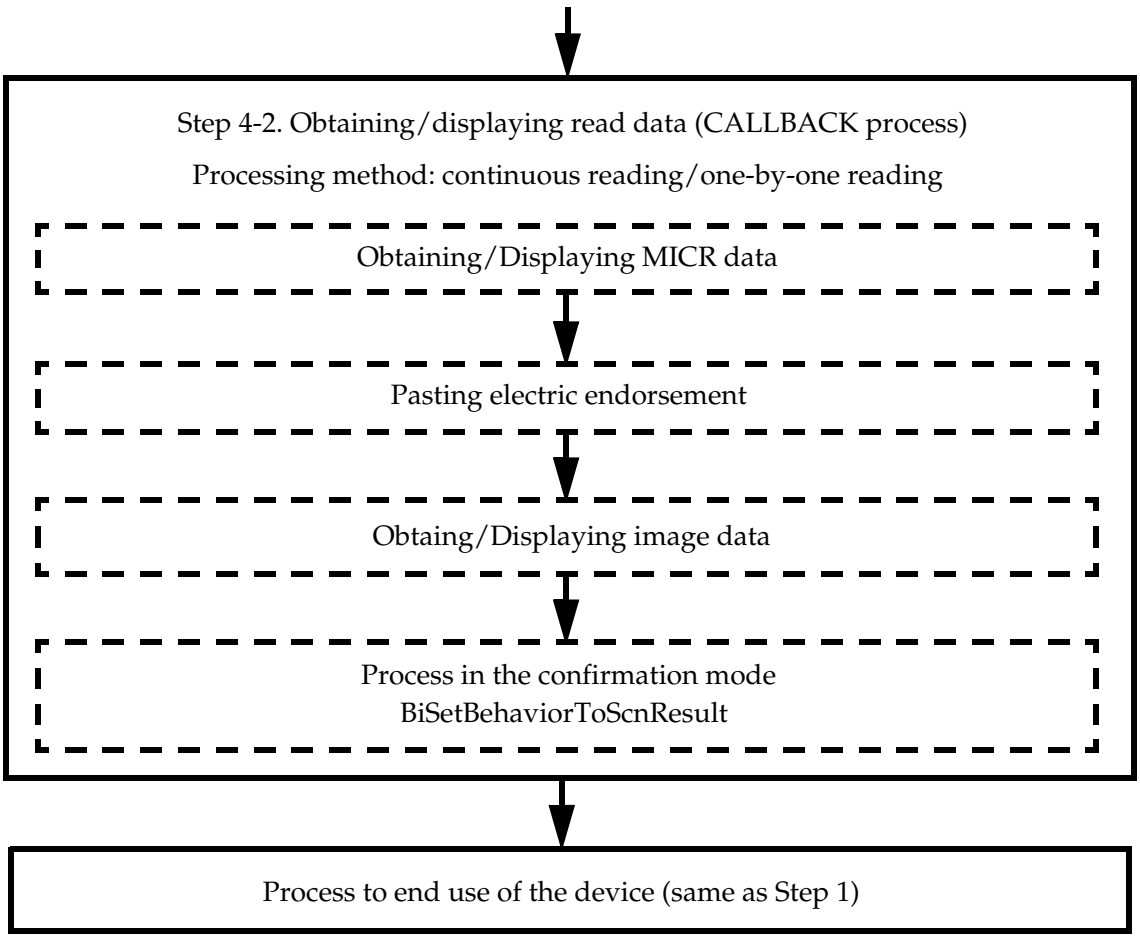
```
CheckResponse(BiSetPrintPosition(m_iHandle, 10, 60));
MF_DECORATE m_tDecorate;
m_tDecorate.dwAttribute = MF_PRINT_COLOR;
m_tDecorate.wFont = MF_PRINT_SYSTEMFONT;
m_tDecorate.szFontName = "Arial";
m_tDecorate.wFontSize = 30;
CheckResponse(BiPrintText(m_iHandle, "Transaction#:<00000000>\x0D\x0A", m_tDecorate));
```

Step 4 Setting the Process When a Reading Error Occurs

In addition to Step 3, sort documents into the two pockets automatically when a reading error occurs or process differently depending on the read result in an application.



* Waterfall mode can also be implemented. For how to implement, see "Step 8 Decoding a barcode, confirming the IQA and Waterfall process" on page 3-61.



Step 4-1. Setting the Process When a Reading Error Occurs

Set the process when a reading error occurs.

Setting MF_PROCESS

Set values for the members of MF_PROCESS with BiSCNMICRFunction before reading process to specify the process mode and process method after errors.

- Calling initial values of the MF_PROCESS
- Changing values of members if necessary
- Setting values of the MP_PROCESS structure

1. Specify the memory address of the MF_PROCESS structure for the second parameter and MF_GET_PROCESS_DEFAULT for the third parameter of BiSCNMICRFunction*XXXX* to call initial values of the MF_PROCESS structure.

   nErr= BiSCNMICRFunctionContinuously (m_iHandle, lpvStruct,
   MF_GET_PROCESS_DEFAULT)

   Note:
   Use BiSCNMICRFunctionPostPrint when the process method is one-by-one reading.

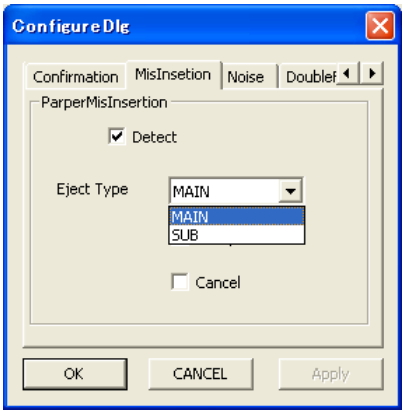
2. Change the default values of members of the MF_PROCESS structure if necessary.

   - Process mode (bActivationMode)

bActivationMode	Description
MF_ACTIVATE_MODE_CONFIRMATION (0)	Processes in the confirmation mode.
MF_ACTIVATE_MODE_HIGH_SPEED (1)	Processes in the high-speed mode.

Note:
In the high-speed mode, documents are automatically processed according to the following MF_PROCESS settings. In the confirmation mode, documents are processed depending on read results by the application's or user's judgement. (See Step 4-2.)

Example:



   - Judgement of insertion orientation error (bPaperMisInsertionErrorSelect)

bPaperMisInsertionErrorSelect	Description
MF_ERROR_SELECT_NODETECT (0)	Does not detect the error.
MF_ERROR_SELECT_DETECT (1)	Detects the error.

   - Ejection pocket when an insertion orientation error (bPaperMisInsertionErrorEject) occurs

bPaperMisInsertionErrorEject	Description
MF_EJECT_MAIN_POCKET (0x22)	Ejects into the main pocket.
MF_EJECT_SUB_POCKET (0x24)	Ejects into the sub pocket.
MF_EJECT_NOEJECT (0x28)	Does not eject.

3. Specify the memory address of the MF_PROCESS structure for the second parameter and MF_SET_PROCESS_PARAM for the third parameter of the BiSCNMICRFunction*XXXX* to set the operation.

   nErr= BiSCNMICRFunctionContinuously (m_iHandle, lpvStruct,
   MF_SET_PROCESS_PARAM)

Note:
Use BiSCNMICRFunctionPostPrint when the process method is one-by-one reading.

Programming code

APIUsage.cpp CAPIUsage::ConfigureMultiple()

APIUsage.cpp CAPIUsage::ConfigureSingle()

```
CheckResponse(BiSCNMICRFunction(m_iHandle, &m_tProcess, MF_GET_PROCESS_DEFAULT));
SetProcessStruct();
CheckResponse(BiSCNMICRFunction(m_iHandle, &m_tProcess, MF_SET_PROCESS_PARAM));
```

APIUsage.cpp CAPIUsage::SetProcessStruct()

```
m_tProcess.bActivationMode = m_pProperties->GetValueDefTrue(ACTIVATE_MODE);

m_tProcess.bPaperMisInsertionErrorSelect = m_pProperties->GetValue-
                                                    DefTrue(PAPER_MIS_INSERT_DETECT);

m_tProcess.bPaperMisInsertionErrorEject = ChangeEject(m_pProperties->GetValueDef-
                                                    False(PAPER_MIS_INSERT_EJECT));
```

Step 4-2. Obtaining/Displaying read data (CALLBACK process)

Documents are automatically processed as set with MF_PROCESS in the high-speed mode, while judgements by applications or users can be added in the confirmation mode. The confirmation mode is described below.

After MF_DATARECEIVE_DONE is confirmed with the CALLBACK function, the next processing method for read data is decided by an application or user. And then, specify the operation of the TM-S1000 with BiSetBehaviorToScnResult. Specify the eject pocket for the second parameter (*bEject*), franking process for the third parameter (*bStamp*), and next reading process for the forth parameter (*bNextCheck*) to process errors for each document.

Note:
When BiSetBehaviorToScnResult is not called before the CALLBACK function is returned, processing is performed following the settings of MF_PROCESS.

nErr = BiSetBehaviorToScnResult (m_iHandle, bEject, bStamp, bNextCheck)

bEject: Specify the ejection pocket

bEject	Description
MF_EJECT_MAIN_POCKET (0x22)	Ejects into the main pocket.
MF_EJECT_SUB_POCKET (0x24)	Ejects into the sub pocket.
MF_EJECT_NOEJECT (0x28)	Does not eject.

bStamp: franking process

bStamp	Description
MF_STAMP_DISABLE (0)	Does not perform franking.
MF_STAMP_ENABLE (1)	Performs franking.

bNextCheck: next reading process

bNextCheck	Description
MF_PROCESS_CONTINUE_OVERLAP (1)	Starts the next reading process while ejecting documents.
MF_PROCESS_CONTINUE_NOOVERLAP (2)	Starts the next reading process after ejecting documents.
MF_PROCESS_CONTINUE_CANCEL (3)	Cancels the next reading process.

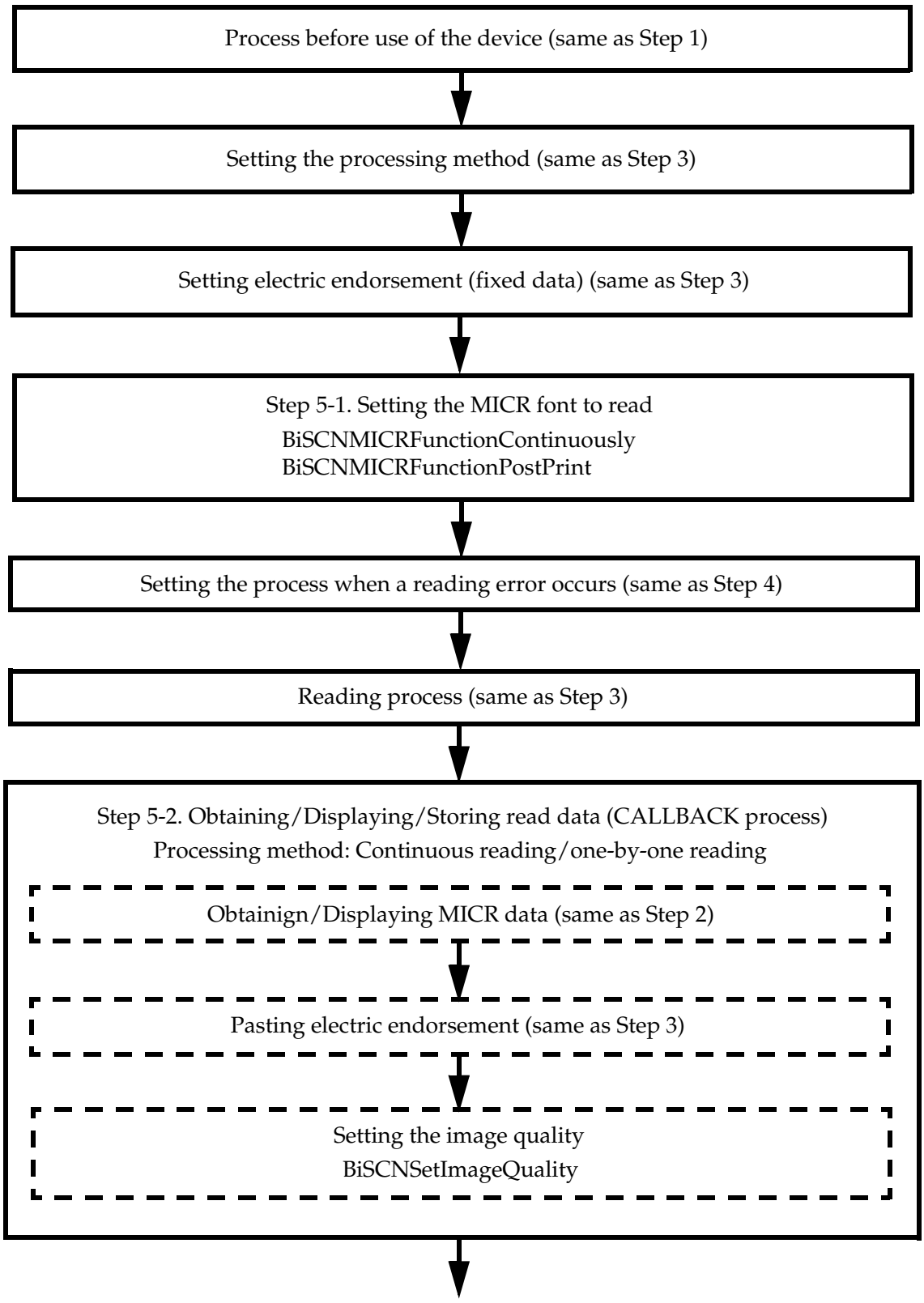
Programming code

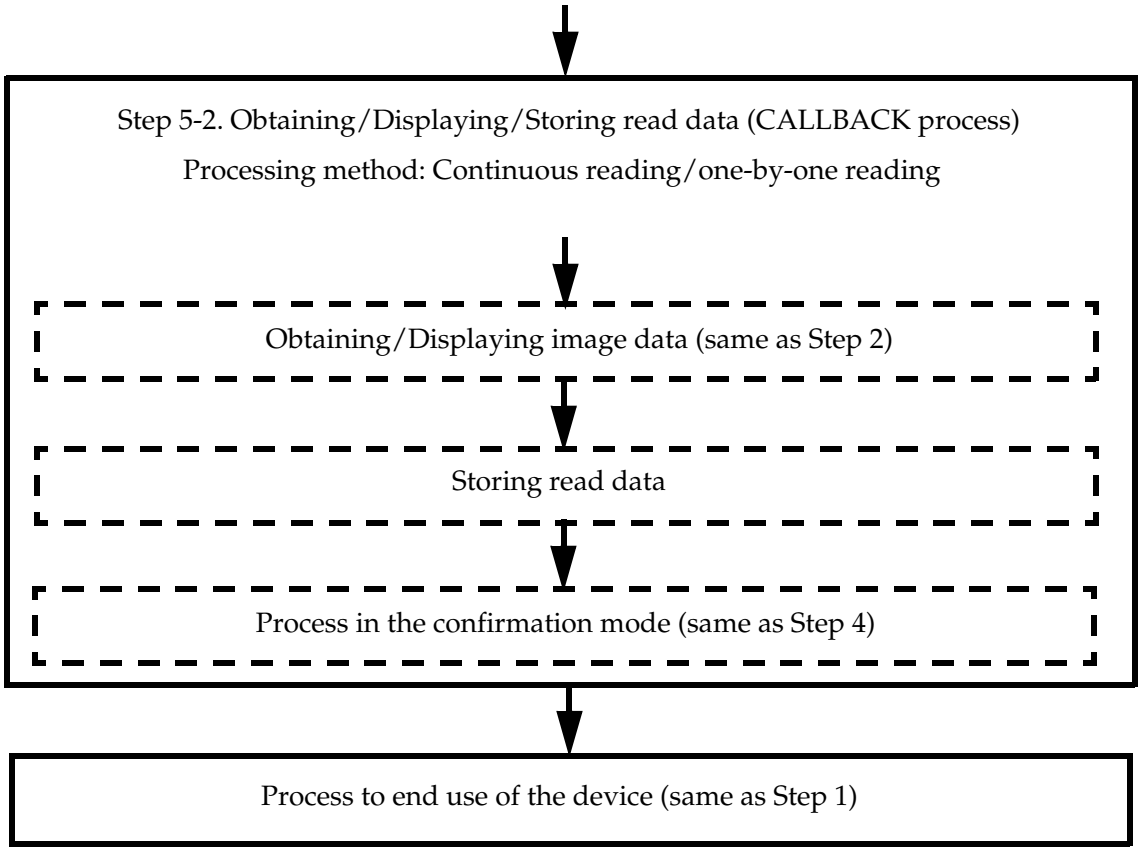
APIUsage.cpp CAPIUsage::ScanStatus()

```
BiSetBehaviorToScnResult(
        m_iHandle,
        ChangeEject(m_pProperties->GetValue(CONFIRMATION_EJECT)),
        m_pProperties->GetValueDefFalse(CONFIRMATION_STAMP),
        ChangeNextCheck(m_pProperties->GetValue(CONFIRMATION_NEXT_CHECK)));
```

Step 5 Setting MICR Font/Image Quality

In addition to the Step 4, select the MICR font and set the image quality.





Step 5-1. Selecting the MICR font

After calling initial values of the MICR structure (MF_MICR), set the MICR font for the member (bFont) of the MICR structure to select the MICR font (E13B or CMC7) to read.

1. Specify the memory address of the MF_MICR structure for the second parameter and MF_GET_MICR_DEFAULT for the third parameter of BiSCNMICRFunction*XXXX* to call initial values of the MICR structure.

   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, lpvStruct, MF_GET_MICR_DEFAULT)

   Note:
   Use BiSCNMICRFunctionContinuously when the process method is continuous reading.

2. Specify the MICR font for bFont, the member of the MF_MICR structure.

bFont	Font
MF_MICR_FONT_E13B (0)	E13B (default)
MF_MICR_FONT_CMC7 (1)	CMC7

3. Specify the memory address of the MF_MICR structure for the second parameter and MF_SET_MICR_PARAM for the third parameter of BiSCNMICRFunction*XXXX* to set the MICR font to read.

   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, lpvStruct, MF_SET_MICR_PARAM)

   Note:
   Use BiSCNMICRFunctionContinuously when the process method is continuous reading.

Programming code

APIUsage.cpp	CAPIUsage::ConfigureMultiple()
APIUsage.cpp	CAPIUsage::ConfigureSingle()

```
CheckResponse(BiSCNMICRFunctionContinuously(m_iHandle, &m_tMicr, MF_GET_MICR_DEFAULT));
if(!m_pProperties->GetValueDefFalse(OCR_FONT)){
        m_tMicr.bFont = MF_MICR_FONT_E13B;
}else{
        m_tMicr.bFont = MF_MICR_FONT_CMC7;
}
CheckResponse(BiSCNMICRFunctionContinuously(m_iHandle, &m_tMicr, MF_SET_MICR_PARAM));
```

Step 5-2. Obtaining/Displaying/Storing read data (CALLBACK process)

After confirming DATA_RECEIVE_DONE with the CALLBACK function, call BiSCNSetimageQuality to set image quality.

Setting scanning format

- Selecting the face to set image quality
- Setting image reading quality

1. Specify the side to set image quality for the second parameter of BiSCNSelectScanFace>

   nErr= BiSCNSelectScanFace(m_iHandle, MF_SCAN_FACE_xxxx)

MF_SCAN_FACE_xxxx	Description
MF_SCAN_FACE_FRONT (0)	Selects the front side
MF_SCAN_FACE_BACK (1)	Selects the back side

2. Set the graduation for the second parameter (*bColorDepth*), threshold value for the third parameter (*bThreshold*), EPS_BI_SCN_MONOCHROME for the forth parameter, and sharpness and so on for the fifth parameter (*ExOption*) of BiSCNSetImageQuality.

   nErr= BiSCNSetImageQuality(m_iHandle, bColorDepth, bThreshold, EPS_BI_SCN_MONOCHROME, bExOption)

   bColorDepth: graduation

bColorDepth	Description
EPS_BI_SCN_1BIT(1)	Black and white (default)
EPS_BI_SCN_8BIT(8)	256 grayscale

   bThreshold: Threshold value

bThreshold	Description
-128 ~ 127	Thicker as the value increases
0	TM-S1000 standard density (default)

   bExOption: Option

bThreshold	Description
EPS_BI_SCN_MANUAL(49)	Thicker as the value increases
EPS_BI_SCN_SHARP(50)	Sharpness (default)

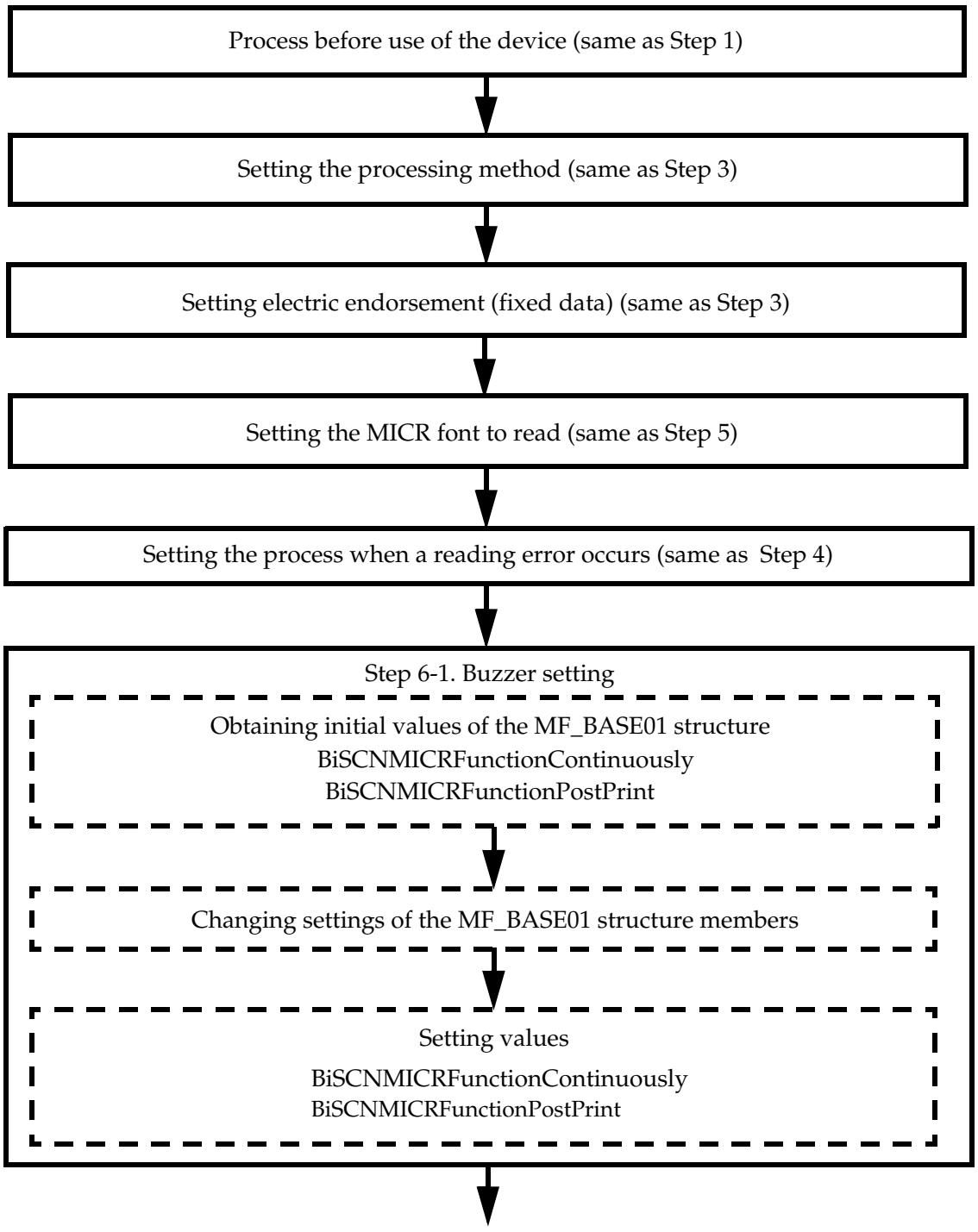
Programming code

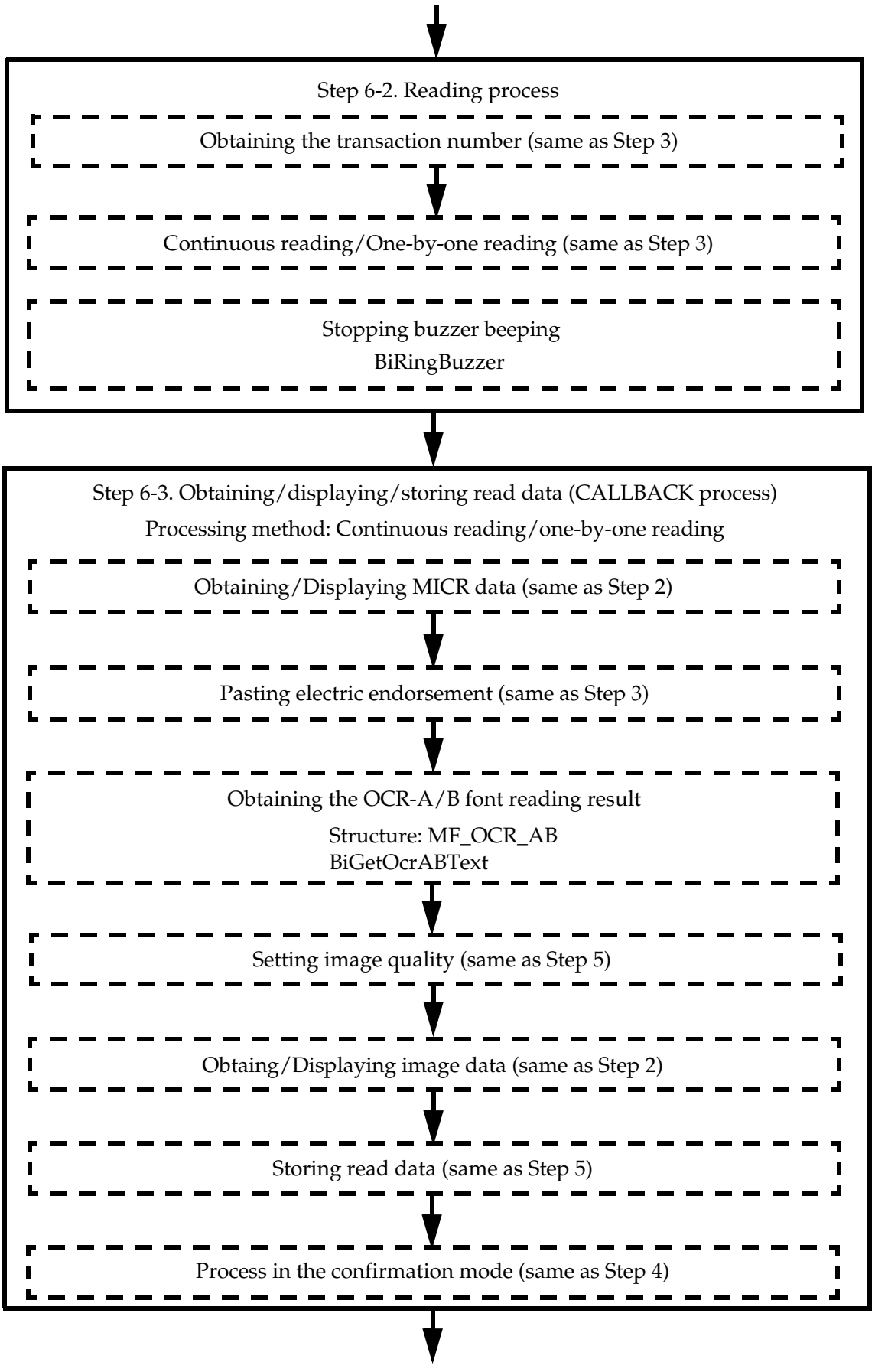
APIUsage.cpp CAPIUsage::ScanStatus()

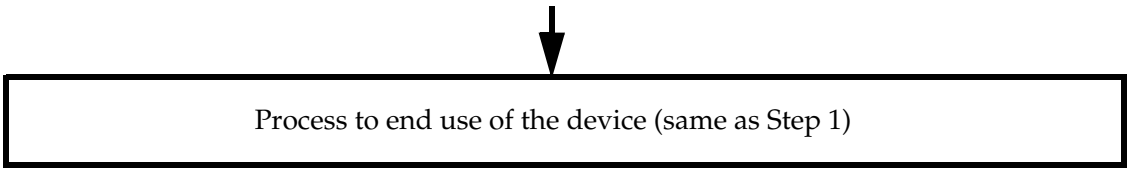
```
iResult = BiSCNSelectScanFace(m_iHandle, MF_SCAN_FACE_FRONT);

if(iResult == SUCCESS){
        BYTE byColorDepth = m_pProperties->GetValueDefFalse(FRONT_GRAYSCALE) ?
                                                EPS_BI_SCN_8BIT : EPS_BI_SCN_1BIT;
        iResult = BiSCNSetImageQuality(m_iHandle, byColorDepth, 0, EPS_BI_SCN_MONOCHROME,
                                        EPS_BI_SCN_SHARP_CUSTOM2);
}
```

Step 6 Reading OCR-A/B Font and Buzzer Setting

In addition to Step 5, read the OCR-A/B font with the OCR function and notify the reading result with the buzzer.




Step 6-2. Reading process
Obtaining the transaction number (same as Step 3)
Continuous reading/One-by-one reading (same as Step 3)
Stopping buzzer beeping
BiRingBuzzer
Step 6-3. Obtaining/displaying/storing read data (CALLBACK process)
Processing method: Continuous reading/one-by-one reading
Obtaining/Displaying MICR data (same as Step 2)
Pasting electric endorsement (same as Step 3)
Obtaining the OCR-A/B font reading result
Structure: MF_OCR_AB
BiGetOcrABText
Setting image quality (same as Step 5)
Obtaing/Displaying image data (same as Step 2)
Storing read data (same as Step 5)
Process in the confirmation mode (same as Step 4)



Step 6-1. Buzzer setting

After calling initial values of the BASE structure (MF_BASE01), set the buzzer frequency for bBuzzerHz (a member of the BASE structure) and the number of buzzer sounds for bBuzzerCount (a member of the BASE structure) to notify reading results with the buzzer sounds. Set the buzzer for each reading result with the array of each BASE structure member.

Reading result	Array
Reading was successful.	MF_BUZZER_TYPE_SUCCESS (0)
Reading error occurred.	MF_BUZZER_TYPE_ERROR (1)
Double-feeding occurred.	MF_BUZZER_TYPE_WFEED (2)

1. Specify the memory address of the MF_BASE01 structure for the second parameter and MF_GET_BASE_DEFAULT for the third parameter of BiSCNMICRFunction*XXXX* to call initial values of the BASE structure.

   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, lpvStruct, MF_GET_BASE_DEFAULT)

   Note:
   Use BiSCNMICRFunctionContinuously when the process method is continuous reading.

   Programming code

   APIUsage.cpp CAPIUsage::ConfigureMultiple()
   APIUsage.cpp CAPIUsage::ConfigureSingle()

```
m_tBase01.iSize = sizeof(MF_BASE01);
m_tBase01.iVersion = MF_BASE_VERSION01;
CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tBase01,
                                                    MF_GET_BASE_DEFAULT)
                                                    );
```

2. Set bBuzzerHz and bBuzzerCount, members of the MF_BASE01 structure as follows:

   bBuzzerHz: frequency

bBuzzerHz	Description
MF_BUZZER_HZ_440 (0)	440Hz
MF_BUZZER_HZ_880 (1)	880Hz
MF_BUZZER_HZ_4000 (2)	4000Hz

bBuzzerCount: number of buzzer sounds

bBuzzerCount	Description
MF_BUZZER_DISABLE (0)	Does not sound the buzzer.
MF_BUZZER_COUNT_MAX (3)	Sounds three times.

Note:
In the sample programs, the number of buzzer sounds is specified by the constant number. Specifying an actual number of buzzer sounds is also possible. (Domain: 0 ≤ bBuzzerCount ≤ 127)

Programming code

APIUsage.cpp CAPIUsage::SetBaseStruct()

```
m_tBase01.bBuzzerHz[MF_BUZZER_TYPE_SUCCESS] = BuzzerHz(m_pProperties->
                                                GetValueDefFalse(BEEP_SUCCESS_HZ));
m_tBase01.bBuzzerCount[MF_BUZZER_TYPE_SUCCESS] = BuzzerCount(m_pProperties->
                                              GetValueDefFalse(BEEP_SUCCESS_COUNT));

m_tBase01.bBuzzerHz[MF_BUZZER_TYPE_ERROR] = BuzzerHz(m_pProperties->
                                                 GetValueDefFalse(BEEP_ERROR_HZ));
m_tBase01.bBuzzerCount[MF_BUZZER_TYPE_ERROR] = BuzzerCount(m_pProperties->
                                               GetValueDefFalse(BEEP_ERROR_COUNT));

m_tBase01.bBuzzerHz[MF_BUZZER_TYPE_WFEED] = BuzzerHz(m_pProperties->
                                                 GetValueDefFalse(BEEP_WFEED_HZ));
m_tBase01.bBuzzerCount[MF_BUZZER_TYPE_WFEED] = BuzzerCount(m_pProperties->
                                               GetValueDefFalse(BEEP_WFEED_COUNT));
```

3. Specify the memory address of the MF_BASE01 structure for the second parameter and MF_SET_BASE_PARAM for the third parameter of BiSCNMICRFunction*XXXX* to set the buzzer.

   nErr= BiSCNMICRFunctionPostPrint (m_iHandle, lpvStruct, MF_SET_BASE_PARAM)

   Note:
   Use BiSCNMICRFunctionContinuously when the process method is continuous reading.

Programming code

APIUsage.cpp CAPIUsage::ConfigureMultiple()
APIUsage.cpp CAPIUsage::ConfigureSingle()

```
CheckResponse(BiSCNMICRFunctionPostPrint(m_iHandle, &m_tBase01, MF_SET_BASE_PARAM));
```

Step 6-2. Reading process

Specify 0 for the second, third, forth, and fifth parameter of BiRingBuzzer while the buzzer is beeping to stop the buzzer.

nErr= BiRingBuzzer (m_iHandle, 0, 0, 0, 0)

Note:
This process stops the beeping sounds, but the buzzer sounds at the next reading process. See "Step 6-1. Buzzer setting" on page 3-46 to disable the buzzer.

Programming code

APIUsage.cpp CAPIUsage::StopBuzzer()

```
BiRingBuzzer(m_iHandle, 0, 0, 0, 0);
```

Step 6-3. Obtaining/Displaying/Storing read data (CALLBACK process)

After confirming DATA_RECEIVE_DONE, set the memory address (MF_OCR_AB) of the OCR_AB structure and call BiGetOcrABText to obtain read results of OCR-A/B from image data.

1. Set values for members of the MF_OCR_AB structure.

   - Fonts to obtain (bOcrType)

bOcrType	Description
MF_OCR_FONT_OCRA_NUM (1)	Numeric OCR-A font
MF_OCR_FONT_OCRA_ALPHA (2)	Alphabetic OCR-A font
MF_OCR_FONT_OCRA_ALPHANUM (3)	Alphanumeric OCR-A font
MF_OCR_FONT_OCRA_ALPHANUM_WOOH (7)	Alphanumeric OCR-A font (except OH)
MF_OCR_FONT_OCRA_ALPHANUM_WOZERO (11)	Alphanumeric OCR-A font (except ZERO)
MF_OCR_FONT_OCRB_NUM (17)	Numeric OCR-B font
MF_OCR_FONT_OCRB_ALPHA (18)	Alphabetic OCR-B font
MF_OCR_FONT_OCRB_ALPHANUM (19)	Alphanumeric OCR-B font
MF_OCR_FONT_OCRB_ALPHANUM_WOOH (23)	Alphanumeric OCR-B font (except OH)
MF_OCR_FONT_OCRB_ALPHANUM_WOZERO (27)	Alphanumeric OCR-B font (except ZERO)

   - Character direction in the OCR readable area (bDirection)

bDirection	Description
MF_OCR_LEFTRIGHT (1)	From left to right (normal direction)
MF_OCR_TOPBOTTOM (2)	From top to bottom (Rotate 90° clockwise)
MF_OCR_RIGHTLEFT (3)	From right to left (upside down)
MF_OCR_BOTTOMTOP (4)	From bottom to top (Rotate 90° counterclockwise)

   - Start position of the OCR readable area in x-coordinate (wStartX)
     Specify in the range from 0 to 255 (unit: mm). If a value out of the range is specified, it is rounded to the nearest value in the range.

- Start position of the OCR readable area in y-coordinate (wStartY)
  Specify in the range from 0 to 255 (unit: mm). If a value out of the range is specified, it is rounded to the nearest value in the range.

- End position of the OCR readable area in x-coordinate (wEndX)
  Specify in the range from 0 to 255 (unit: mm). If a value out of the range is specified, it is rounded to the nearest value in the range.
  If OCR_AREA_RIGHT is specified, the right end of image data is specified.

- End position of the OCR readable area in y-coordinate (wEndY)
  Specify in the range from 0 to 255 (unit: mm). If a value out of the range is specified, it is rounded to the nearest value in the range.
  If OCR_AREA_BOTTOM is specified, the bottom end of image data is specified.

- Handling space characters (bSpaceHandling)

bSpaceHandling	Description
OCR_SPACE_ENABLE (1)	Includes space characters in the OCR reading results.
OCR_SPACE_DISABLE (0)	Does not include space characters in the OCR reading results.

2. Specify the transaction number (dwTransactionNumber) for the second parameter, OCR_SOURCE_TRANSACTION_NUMBER for the third parameter (*bImageSource*), "" for the forth parameter (*szFileName*), and the memory address of the OCR_AB structure for the fifth parameter (*ptMicr*) of BiGetOcrABText to obtain OCR-A/B font data.

   nErr = BiGetOcrABText (m_iHandle , dwTransactionNumber ,
   OCR_SOURCE_TRANSACTION_NUMBER , “” ,
   MF_OCR_AB)

3. OCR-A/B font data returns to szOcrStr of the MF_OCR_AB structure.

4. Obtain OCR-A/B font data in the CALLBACK function and display it on the application.

Programming code

APIUsage.cpp CAPIUsage::ScanStatus()

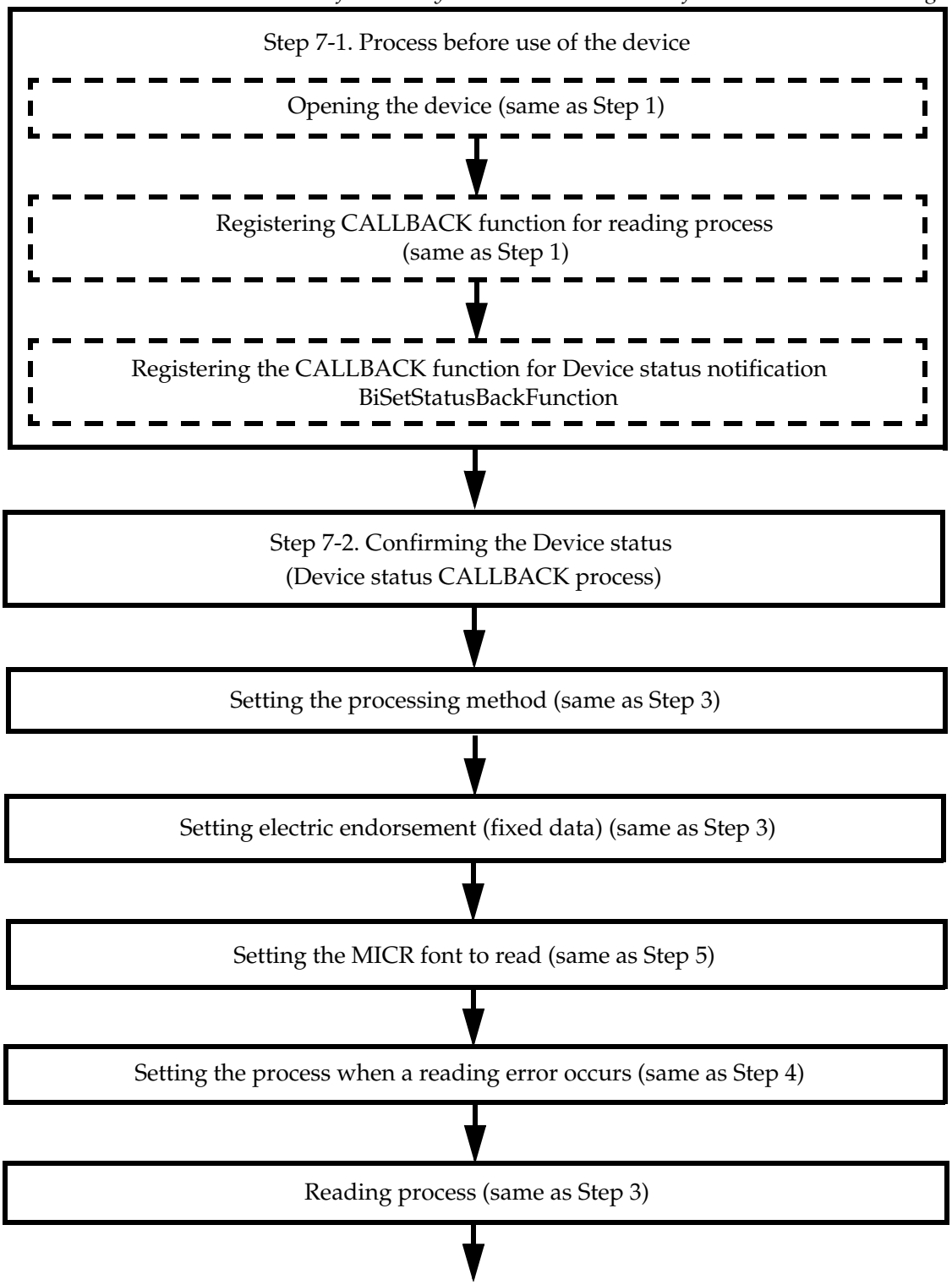
```
// get ocr ab data
TCHAR pOcrAb[1024];
MF_OCR_AB mfOcrAb;
mfOcrAb.iSize = sizeof(MF_OCR_AB);
mfOcrAb.iVersion = MF_OCR_AB_VERSION;
//Specify Font
switch(m_pProperties->GetValueDefFalse(OCR_AB_FONT)){
        case 0:
                mfOcrAb.bOcrType = MF_OCR_FONT_OCRA_ALPHANUM;
                _tcscpy(pOcrAb, "OCR_A:");
                break;
        case 1:
                mfOcrAb.bOcrType = MF_OCR_FONT_OCRB_ALPHANUM;
                _tcscpy(pOcrAb, "OCR_B:");
                break;
}
```

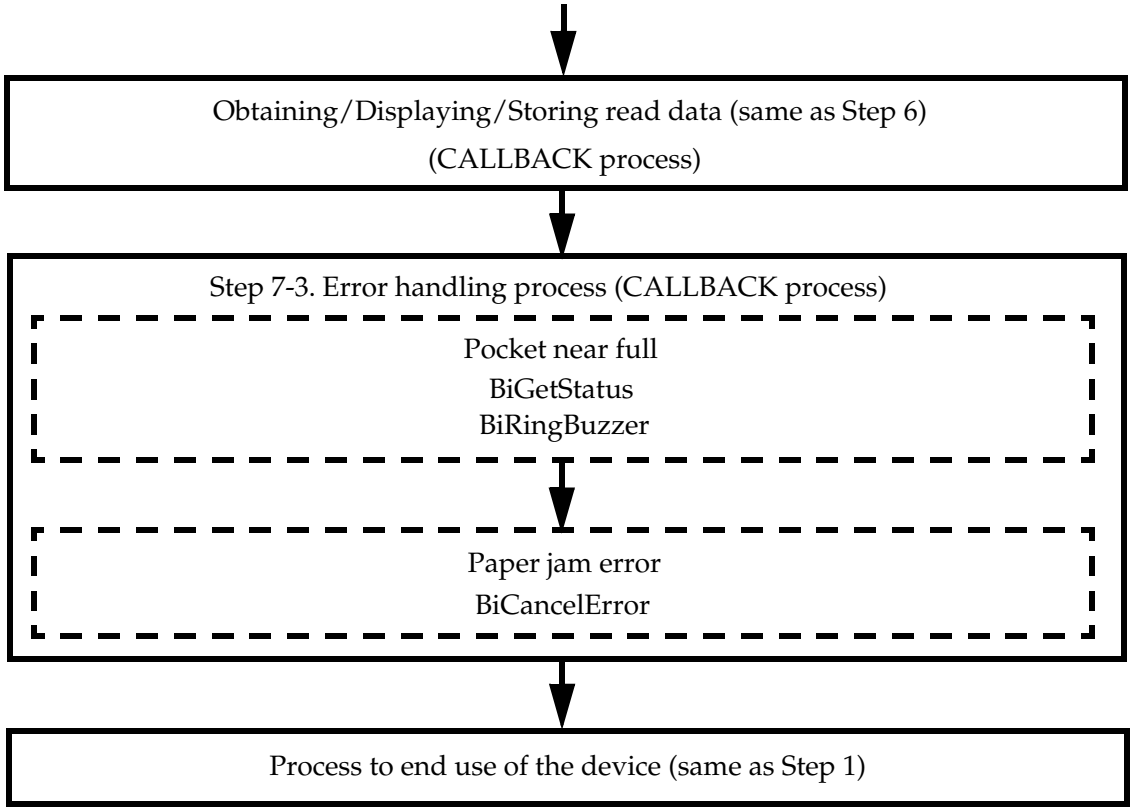
```
mfOcrAb.bDirection = MF_OCR_LEFTRIGHT;
mfOcrAb.wStartX = 30;
mfOcrAb.wStartY = 40;
mfOcrAb.wEndX = 110;
mfOcrAb.wEndY = 75;
mfOcrAb.bSpaceHandling = OCR_SPACE_ENABLE;
::ZeroMemory(mfOcrAb.szOcrStr, sizeof(mfOcrAb.szOcrStr));
//Obtain the OCR_AB character string.
iResult = BiGetOcrABText(m_iHandle, dwTransactionNumber, OCR_SOURCE_TRANSACTION_NUMBER,
                                                                                    "", &mfOcrAb);
//Copy the obtained character string.
_tcscpy(pOcrAb + strlen("OCR_X:"), mfOcrAb.szOcrStr);
if(iResult != SUCCESS){
        //Add a error code.
        _tcscat(pOcrAb, " ( ");
        _tcscat(pOcrAb, GetResultString(iResult));
        _tcscat(pOcrAb, " ) ");
}
//Display the OCR_AB character string.
pDlg->SetOcrString(pOcrAb);
```

Step 7 Confirming the Device status and error handling

In addition to Step 6, describes how to confirm the device status, how to handle errors (pocket near-full/paper jam error), and how to process MICR cleaning.

Note:
Device status is a status that is notified mainly when the sensor status of the TM-S1000 has changed.





Step 7-1. Process before use of the device

Registering the CALLBACK function for Device status notification

Register the CALLBACK function with BiSetStatusBackFunction, then the event (Device status) that occurs mainly when the sensor status of the TM-S1000 has changed calls the CALLBACK function. Specify the CALLBACK function name for the second parameter of BiSetStatusBackFunction to register the CALLBACK function.

nErr= BiSetStatusBackFunction (m_iHandle , cbStatus)

Note:

In sample programs, the CALLBACK function named cbStatus is specified. The notified status (dwStatus) can be obtained in this CALLBACK function.

Programming code

APIUsage.cpp

```
// Called when notifying the status
int CALLBACK cbStatus(DWORD dwStatus)
{
        // Sets the status in DeviceStatusDlg
        ((CTMS1000SampleDlg*)(theApp.m_pMainWnd))->m_DeviceStatusDlg.SetDeviceStatus(dwStatus);

        return 0;
}
```

APIUsage.cpp CAPIUsage::CAPIUsage()

```
// Register the callback destination of the status notification
CheckResponse(BiSetStatusBackFunction(m_iHandle, cbStatus));
```

Step 7-2. Confirming Device status

When the status of the device (such as sensors) changes, the CALLBACK function registered in "Step 7-1. Process before use of the device" is called back. The device status can be confirmed with the notified Device status (*dwStatus*).) For types of Device status, see "Device Status" on page 4-1.

Note:
In sample programs, Device status is notified to dwStatus, a parameter.
Registering the CALLBACK function is not necessary to obtain the Device status. By calling BiGetStatus, it can be obtained in real time even during the reading process. (See "Step 7-3. Error handling" on page 3-55.) *All sensor statuses detected by the TM-S1000 are included in the obtained status (4 bytes). For the sensors, see the TM-S1000 Technical Reference Guide.*

Programming code

DeviceStatusDlg.cpp CDeviceStatusDlg::SetDeviceStatus(DWORD dwStatus)

```
// Renewal the check box by the status
void CDeviceStatusDlg::SetDeviceStatus(DWORD dwStatus)
{
        // Display the status
        CString strStatus;
        strStatus.Format("Device Status - %08Xh", dwStatus);
        ::SetWindowText(this->m_hWnd, strStatus);

        // ASB_NO_RESPONSE
        if(dwStatus == ASB_NO_RESPONSE){
                // ASB_NO_RESPONSE
                ((CButton*)GetDlgItem(IDC_CHECK_NO_RESPONSE))->SetCheck(TRUE);
                // ASB_OFF_LINE
                ((CButton*)GetDlgItem(IDC_CHECK_OFF_LINE))->SetCheck(FALSE);
                // ASB_COVER_OPEN
                ((CButton*)GetDlgItem(IDC_CHECK_COVER_OPEN))->SetCheck(FALSE);
                // ASB_WAIT_PEPRT_EJECT
                ((CButton*)GetDlgItem(IDC_CHECK_WAIT_PEPRT_EJECT))->SetCheck(FALSE);
                // ASB_MECHANICAL_ERR
                ((CButton*)GetDlgItem(IDC_CHECK_MECHANICAL_ERR))->SetCheck(FALSE);
                // ASB_UNRECOVER_ERR
                ((CButton*)GetDlgItem(IDC_CHECK_UNRECOVER_ERR))->SetCheck(FALSE);
                // ASB_PAPER_INTERMEDIATE
                ((CButton*)GetDlgItem(IDC_CHECK_PAPER_INTERMEDIATE))->SetCheck(FALSE);
                // ASB_MAIN_NEAR_FULL
                ((CButton*)GetDlgItem(IDC_CHECK_MAIN_NEAR_FULL))->SetCheck(FALSE);
                // ASB_EJECT_SENSOR_NO_PAPER
                ((CButton*)GetDlgItem(IDC_CHECK_EJECT_SENSOR_NO_PAPER))->
                                                                SetCheck(FALSE);
                // ASB_SUB_NEAR_FULL
                ((CButton*)GetDlgItem(IDC_CHECK_SUB_NEAR_FULL))->SetCheck(FALSE);
                // ASB_SLIP_PAPER_SIZE
                ((CButton*)GetDlgItem(IDC_CHECK_SLIP_PAPER_SIZE))->SetCheck(FALSE);
                // ASB_ASF_PAPER
                ((CButton*)GetDlgItem(IDC_CHECK_ASF_PAPER))->SetCheck(FALSE);
                // ASB_STAMP_EXIST
                ((CButton*)GetDlgItem(IDC_CHECK_STAMP_EXIST))->SetCheck(FALSE);
                // ASB_WAIT_INSERT
```

```
                ((CButton*)GetDlgItem(IDC_CHECK_WAIT_INSERT))->SetCheck(FALSE);
                // ASB_FRANKING_SENSOR
                ((CButton*)GetDlgItem(IDC_CHECK_FRANKING_SENSOR))->SetCheck(FALSE);
        }else{
                // Not ASB_NO_RESPONSE
                // ASB_NO_RESPONSE
                ((CButton*)GetDlgItem(IDC_CHECK_NO_RESPONSE))->SetCheck(FALSE);
                // ASB_OFF_LINE
                ((CButton*)GetDlgItem(IDC_CHECK_OFF_LINE))->
                                        SetCheck(GetEnable(dwStatus & ASB_OFF_LINE));
                // ASB_COVER_OPEN
                ((CButton*)GetDlgItem(IDC_CHECK_COVER_OPEN))->
                                SetCheck(GetEnable(dwStatus & ASB_COVER_OPEN));
                // ASB_WAIT_PEPRT_EJECT
                ((CButton*)GetDlgItem(IDC_CHECK_WAIT_PEPRT_EJECT))->
                        SetCheck(GetEnable(dwStatus & ASB_WAIT_PEPRT_EJECT));
                // ASB_MECHANICAL_ERR
                ((CButton*)GetDlgItem(IDC_CHECK_MECHANICAL_ERR))->
                                SetCheck(GetEnable(dwStatus & ASB_MECHANICAL_ERR));
                // ASB_UNRECOVER_ERR
                ((CButton*)GetDlgItem(IDC_CHECK_UNRECOVER_ERR))->
                                SetCheck(GetEnable(dwStatus & ASB_UNRECOVER_ERR));
                // ASB_PAPER_INTERMEDIATE
                ((CButton*)GetDlgItem(IDC_CHECK_PAPER_INTERMEDIATE))->
                        SetCheck(!GetEnable(dwStatus & ASB_PAPER_INTERMEDIATE));
                // ASB_MAIN_NEAR_FULL
                ((CButton*)GetDlgItem(IDC_CHECK_MAIN_NEAR_FULL))->
                                SetCheck(!GetEnable(dwStatus & ASB_MAIN_NEAR_FULL));
                // ASB_EJECT_SENSOR_NO_PAPER
                ((CButton*)GetDlgItem(IDC_CHECK_EJECT_SENSOR_NO_PAPER))->
                        SetCheck(!GetEnable(dwStatus & ASB_EJECT_SENSOR_NO_PAPER));
                // ASB_SUB_NEAR_FULL
                ((CButton*)GetDlgItem(IDC_CHECK_SUB_NEAR_FULL))->
                                SetCheck(!GetEnable(dwStatus & ASB_SUB_NEAR_FULL));
                // ASB_SLIP_PAPER_SIZE
                ((CButton*)GetDlgItem(IDC_CHECK_SLIP_PAPER_SIZE))->
                                SetCheck(!GetEnable(dwStatus & ASB_SLIP_PAPER_SIZE));
                // ASB_ASF_PAPER
                ((CButton*)GetDlgItem(IDC_CHECK_ASF_PAPER))->
                                SetCheck(!GetEnable(dwStatus & ASB_ASF_PAPER));
                // ASB_STAMP_EXIST
                ((CButton*)GetDlgItem(IDC_CHECK_STAMP_EXIST))->
                                SetCheck(GetEnable(dwStatus & ASB_STAMP_EXIST));
                // ASB_WAIT_INSERT
                ((CButton*)GetDlgItem(IDC_CHECK_WAIT_INSERT))->
                                SetCheck(GetEnable(dwStatus & ASB_WAIT_INSERT));
                // ASB_FRANKING_SENSOR
                ((CButton*)GetDlgItem(IDC_CHECK_FRANKING_SENSOR))->
                        SetCheck(!GetEnable(dwStatus & ASB_FRANKING_SENSOR));
        }
}
```

Step 7-3. Error handling

Errors that occur in the reading process are handled within the CALLBACK function of the reading process. The sample program executes the process of pocket near full and paper jam error. For other error handling, see the sub status of the CALLBACK function of reading process (See "BiSCNMICRFunctionContinuously" on page 4-58) or device status (See "Device Status" on page 4-1).

Pocket near full

In the sample program, a pocket near full is handled in the following steps.

1. Confirms the status of MF_CHECKPAPER_PROCESS_DONE with the CALLBACK function.

2. Executes BiGetStatus to obtain the device status. The device status returns to the second parameter (*lpStatus*) of BiGetStatus.

   BiGetStatus (m_iHandle, lpStatus)

3. Confirms the pocket near full with the obtained device status (*lpStatus*). For details on Device status, see "Device Status" on page 4-1.

4. In case of the pocket near full, notifies users. In the sample program, displays a message or beeps the buzzer (BiRingBuzzer) to notify users of the pocket near full.

Programming code

APIUsage.cpp CAPIUsage::ScanStatus()

```
if(wMainStatus == MF_CHECKPAPER_PROCESS_DONE){
        DWORD dwStatus = GetStatus();
        if(!m_bNearFullMsg){
                // Near full error
                if(!(dwStatus & ASB_MAIN_NEAR_FULL) || !(dwStatus & ASB_SUB_NEAR_FULL)){
                        m_bNearFullMsg = TRUE;
                        pDlg->SetWindowText("TM-S1000SampleStep7  -  ***** NearFull *****");
                        BiRingBuzzer(m_iHandle, MF_BUZZER_TONE_HIGH, 3, 1, 0);
                }
        }else{
                // No Near full error
                if((dwStatus & ASB_MAIN_NEAR_FULL) && (dwStatus & ASB_SUB_NEAR_FULL)){
                        m_bNearFullMsg = FALSE;
                        pDlg->SetWindowText("TM-S1000SampleStep7");
                }
        }
}
```

Paper jam error

In the sample program, the following paper jam errors are handled.

- When a paper jam error occurs and reading process ends.
- When a paper jam error occurs and reading process continues.

For the handling timing of the paper jam error, see "9. CALLBACK process (paper jam error occurred)" on page 3-6.

❏ Error handling when a paper jam error occurs and reading process ends

1. Confirms the status of MF_FUNCTION_DONE with the CALLBACK function to confirm that the reading process has ended.
2. With the sub status (wSubStatus) of the CALLBACK function, confirms that the reading process has ended normally. For details, see "MF_BASE.iRet" on page 4-62.
3. If ERR_PAPER_JAM is confirmed for the sub status, a paper jam error has occurred. Displays a message to invoke users to remove the paper in the paper path.
4. After confirming that the paper in the paper path has been removed, call BiCancelError to cancel the error.

BiCancelError (m_iHandle)

Programming code

APIUsage.cpp CAPIUsage::ScanStatus()

```
if(wMainStatus == MF_FUNCTION_DONE){
        ::SetEvent(m_hScanEvent);
        if((short)wSubStatus == ERR_PAPER_JAM){
                // Display the prompt "Remove the paper".
                ::MessageBox(pDlg->m_hWnd, "Remove the paper", "MECHANICAL ERROR",
                                                                        MB_OK);
                // Cancel mechanical error
                BiCancelError(m_iHandle);
        }else if(wSubStatus != SUCCESS){
                ::MessageBox(pDlg->m_hWnd, GetResultString((short)wSubStatus), "Error", 0);
        }
}
```

❑ Error handling when a paper jam error occurs and reading process continues

Note:
Set MF_RESULT_PARTIAL for bResultPartialData of the MF_PROCESS structure. If the reading process does not continue, set MF_RESULT_NONE. In that case, a paper jam error is handled with the former method.

1. Confirms the status of MF_ERROR_OCCURED with the CALLBACK function when an error occurs.
2. With the sub status (wSubStatus) of the CALLBACK function, confirms the error content. For details, see "Processing status list" on page 4-65.
3. If ERR_PAPER_JAM is confirmed for the sub status, a paper jam error has occurred. Displays a message to inform users of the paper jam. The reading process continues.
4. After that, confirms the status of MF_FUNCTION_DONE with the CALLBACK function to confirm that the reading process has ended.
5. Confirms ERR_PAPER_JAM for the sub status of the CALLBACK function and displays a message to invoke users to remove the paper in the paper path.
6. After confirming that the paper in the paper path has been removed, call BiCancelError to cancel the error.

BiCancelError (m_iHandle)

Programming code

APIUsage.cpp CAPIUsage::ScanStatus()

```
if(wMainStatus == MF_ERROR_OCCURED){
        m_wErrorOccured = wSubStatus;
}

if(wMainStatus == MF_FUNCTION_DONE){
        ::SetEvent(m_hScanEvent);
        if((short)wSubStatus == ERR_PAPER_JAM){
                // Display the prompt "Remove the paper".
                ::MessageBox(pDlg->m_hWnd, "Remove the paper", "MECHANICAL ERROR",
                                                                        MB_OK);
                // Cancel mechanical error
                BiCancelError(m_iHandle);
        }else if(wSubStatus != SUCCESS){
                ::MessageBox(pDlg->m_hWnd, GetResultString((short)wSubStatus), "Error", 0);
        }
}
```

Conf.cpp CConf::OnInitDialog()

```
SetWindowText(((CTMS1000SampleDlg*)theApp.m_pMainWnd)->m_api.GetErrorOccured());
```

APIUsage.cpp LPCTSTR CAPIUsage::GetErrorOccured()

```
LPCTSTR CAPIUsage::GetErrorOccured()
{
	return GetResultString((short)m_wErrorOccured);
}
```

APIUsage.h static LPCTSTR GetResultString(int iResultCode)

```
static LPCTSTR GetResultString(int iResultCode) {
	switch(iResultCode) {
		case SUCCESS:
			return _T("SUCCESS");
		case ERR_TYPE:
			return _T("ERR_TYPE");
		case ERR_OPENED:
			return _T("ERR_OPENED");
		case ERR_NO_PRINTER:
			return _T("ERR_NO_PRINTER");
		case ERR_NO_TARGET:
			return _T("ERR_NO_TARGET");
		case ERR_NO_MEMORY:
			return _T("ERR_NO_MEMORY");
		case ERR_HANDLE:
			return _T("ERR_HANDLE");
		case ERR_TIMEOUT:
			return _T("ERR_TIMEOUT");
		case ERR_ACCESS:
			return _T("ERR_ACCESS");
		case ERR_PARAM:
			return _T("ERR_PARAM");
		case ERR_NOT_SUPPORT:
			return _T("ERR_NOT_SUPPORT");
		case ERR_OFFLINE:
			return _T("ERR_OFFLINE");
		case ERR_NOT_EPSON:
			return _T("ERR_NOT_EPSON");
		case ERR_WITHOUT_CB:
			return _T("ERR_WITHOUT_CB");
		case ERR_BUFFER_OVER_FLOW:
			return _T("ERR_BUFFER_OVER_FLOW");
		case ERR_REGISTRY:
			return _T("ERR_REGISTRY");
		case ERR_ENABLE:
			return _T("ERR_ENABLE");
		case ERR_DISK_FULL:
			return _T("ERR_DISK_FULL");
		case ERR_NO_IMAGE:
			return _T("ERR_NO_IMAGE");
		case ERR_ENTRY_OVER:
			return _T("ERR_ENTRY_OVER");
		case ERR_CROPAREAID:
			return _T("ERR_CROPAREAID");
		case ERR_EXIST:
			return _T("ERR_EXIST");
		case ERR_NOT_FOUND:
			return _T("ERR_NOT_FOUND");
```

```
case ERR_IMAGE_FILEOPEN:
        return _T("ERR_IMAGE_FILEOPEN");
case ERR_IMAGE_UNKNOWNFORMAT:
        return _T("ERR_IMAGE_UNKNOWNFORMAT");
case ERR_IMAGE_FAILED:
        return _T("ERR_IMAGE_FAILED");
case ERR_WORKAREA_NO_MEMORY:
        return _T("ERR_WORKAREA_NO_MEMORY");
case ERR_WORKAREA_UNKNOWNFORMAT:
        return _T("ERR_WORKAREA_UNKNOWNFORMAT");
case ERR_WORKAREA_FAILED:
        return _T("ERR_WORKAREA_FAILED");
case ERR_IMAGE_FILEREAD:
        return _T("ERR_IMAGE_FILEREAD");
case ERR_PAPERINSERT_TIMEOUT:
        return _T("ERR_PAPERINSERT_TIMEOUT");
case ERR_EXEC_FUNCTION:
        return _T("ERR_EXEC_FUNCTION");
case ERR_EXEC_MICR:
        return _T("ERR_EXEC_MICR");
case ERR_EXEC_SCAN:
        return _T("ERR_EXEC_SCAN");
case ERR_SS_NOT_EXIST:
        return _T("ERR_SS_NOT_EXIST");
case ERR_SPL_NOT_EXIST:
        return _T("ERR_SPL_NOT_EXIST");
case ERR_SPL_PAUSED:
        return _T("ERR_SPL_PAUSED");
case ERR_RESET:
        return _T("ERR_RESET");
case ERR_THREAD:
        return _T("ERR_THREAD");
case ERR_ABORT:
        return _T("ERR_ABORT");
case ERR_MICR:
        return _T("ERR_MICR");
case ERR_SCAN:
        return _T("ERR_SCAN");
case ERR_LINE_OVERFLOW:
        return _T("ERR_LINE_OVERFLOW");
case ERR_NOT_EXEC:
        return _T("ERR_NOT_EXEC");
case ERR_SIZE:
        return _T("ERR_SIZE");
case ERR_PAPER_PILED:
        return _T("ERR_PAPER_PILED");
case ERR_PAPER_JAM:
        return _T("ERR_PAPER_JAM");
case ERR_COVER_OPEN:
        return _T("ERR_COVER_OPEN");
case ERR_MICR_NODATA:
        return _T("ERR_MICR_NODATA");
case ERR_MICR_BADDATA:
        return _T("ERR_MICR_BADDATA");
```

```
                    case ERR_MICR_PARSE:
                        return _T("ERR_MICR_PARSE");
                    case ERR_MICR_NOISE:
                        return _T("ERR_MICR_NOISE");
                    case ERR_SCN_COMPRESS:
                        return _T("ERR_SCN_COMPRESS");
                    case ERR_PAPER_EXIST:
                        return _T("ERR_PAPER_EXIST");
                    case ERR_PAPER_INSERT:
                        return _T("ERR_PAPER_INSERT");
                    case ERR_EXEC_SCAN_CHECK_CONTINUOUS:
                        return _T("ERR_EXEC_SCAN_CHECK_CONTINUOUS");
                    case ERR_EXEC_SCAN_CHECK_ONEBYONE:
                        return _T("ERR_EXEC_SCAN_CHECK_ONEBYONE");
                    case ERR_EXEC_SCAN_IDCARD:
                        return _T("ERR_EXEC_SCAN_IDCARD");
                    case ERR_EXEC_PRINT_ROLLPAPER:
                        return _T("ERR_EXEC_PRINT_ROLLPAPER");
                    case ERR_EXEC_PRINT_VALIDATION:
                        return _T("ERR_EXEC_PRINT_VALIDATION");
                }
                static CString strResult;
                strResult.Format("%d", iResultCode);
                return strResult;
            }
```

Step 7-5. Cleaning the MICR mechanism

Load a cleaning sheet in the ASF, and then call BiMICRCleaning to clean the MICR mechanism. (For the cleaning sheet, see the "TM-S1000 User's Manual" or the "TM-S1000 Technical Reference Guide.")

nErr= BiMICRCleaning (m_iHandle)

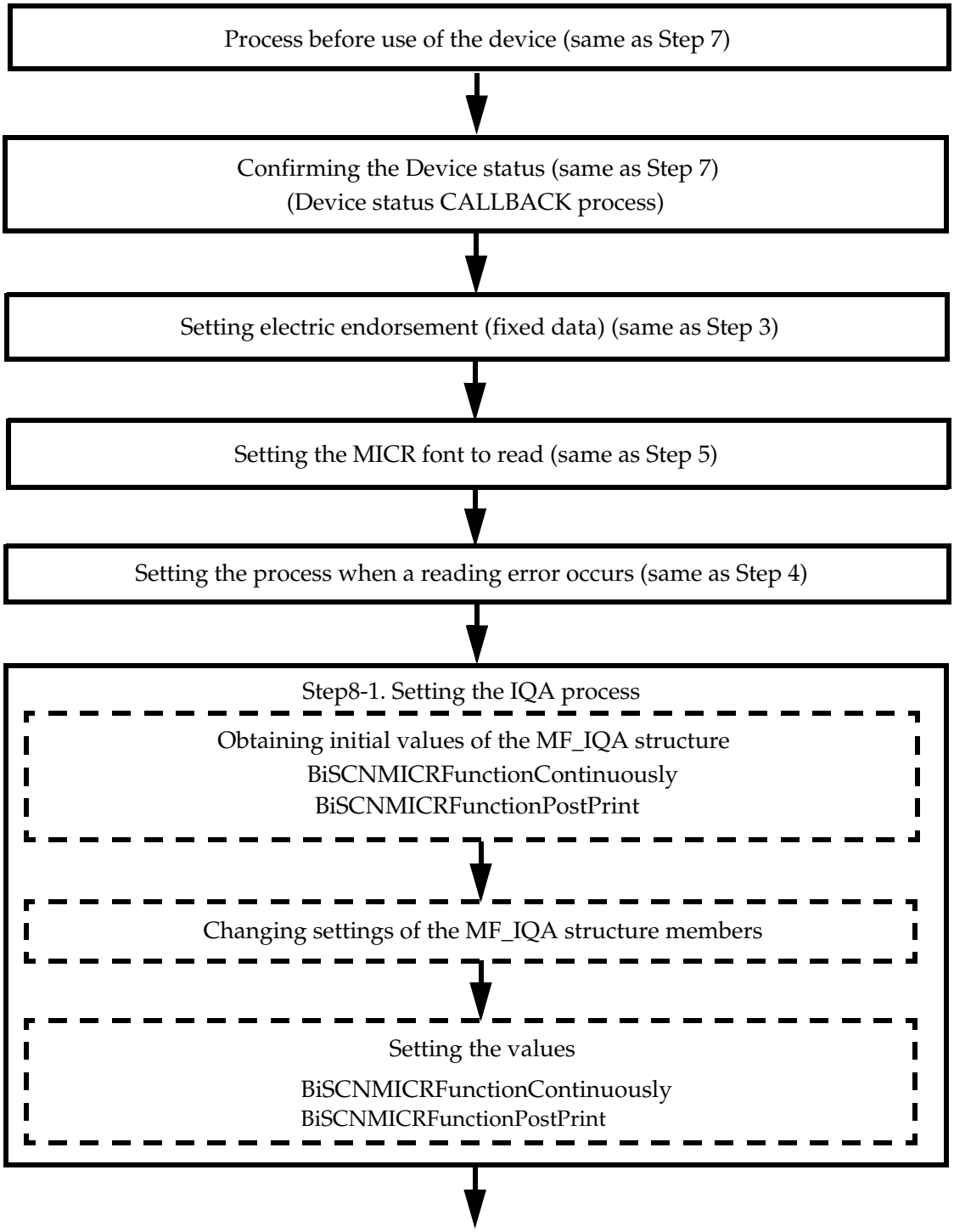
Programming code

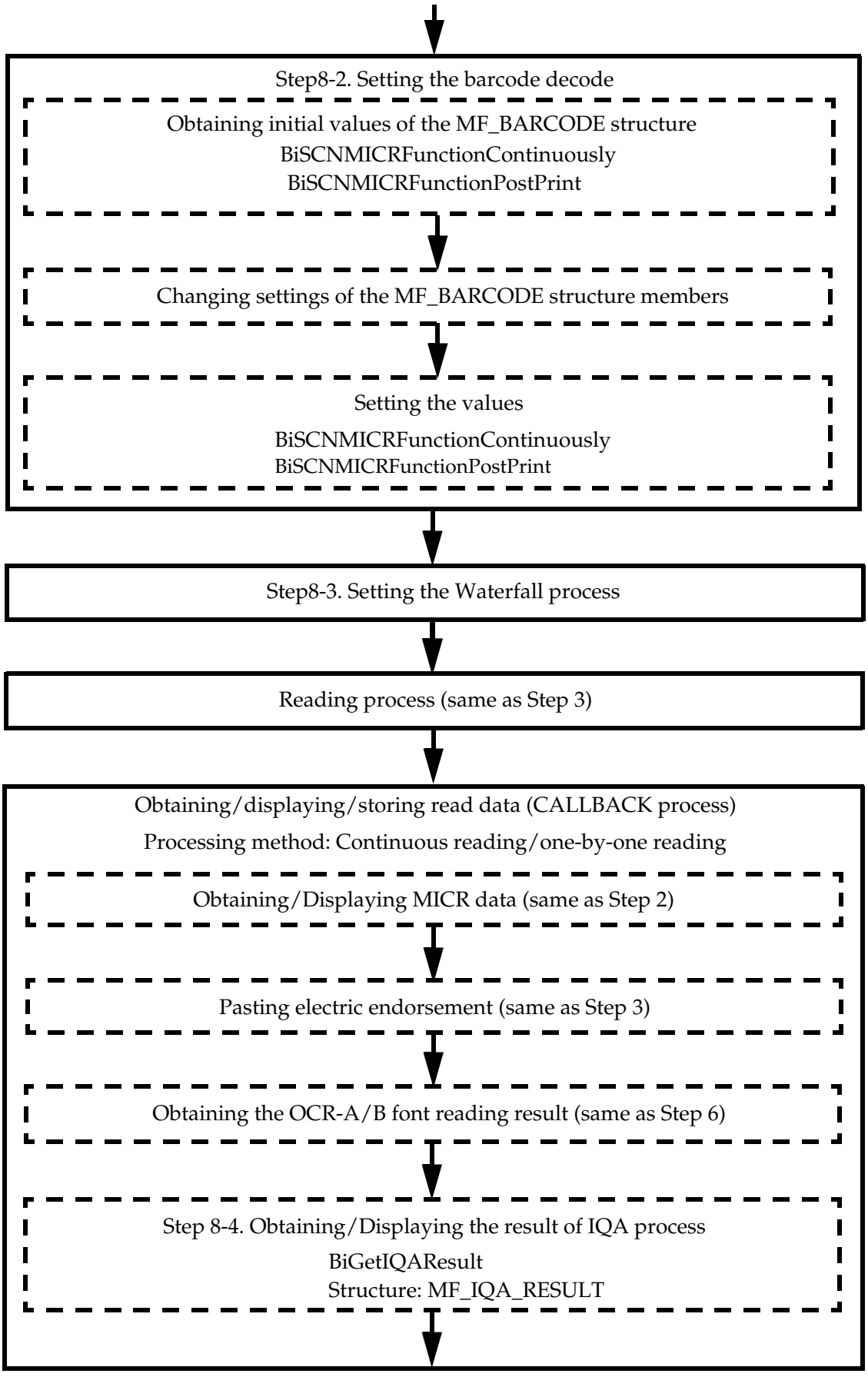
APIUsage.cpp CAPIUsage::Cleaning()

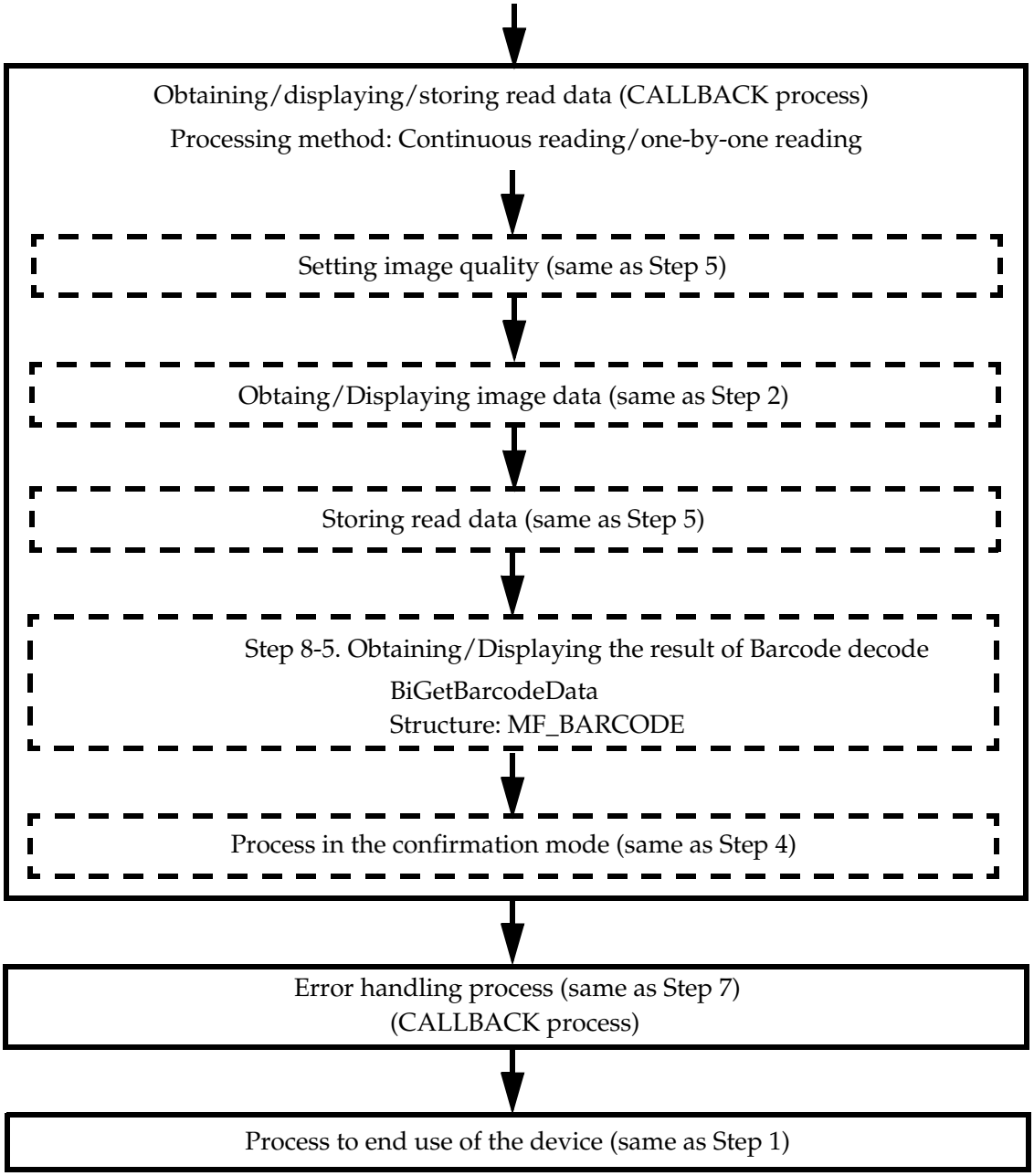
```
CheckResponse(BiMICRCleaning(m_iHandle));
```

Step 8 Decoding a barcode, confirming the IQA and Waterfall process

In addition to Step 7, describes how to decode a barcode, how to confirm the IQA, and how to process Waterfall.




Step8-2. Setting the barcode decode
Obtaining initial values of the MF_BARCODE structure
BiSCNMICRFunctionContinuously
BiSCNMICRFunctionPostPrint
Changing settings of the MF_BARCODE structure members
Setting the values
BiSCNMICRFunctionContinuously
BiSCNMICRFunctionPostPrint
Step8-3. Setting the Waterfall process
Reading process (same as Step 3)
Obtaining/displaying/storing read data (CALLBACK process)
Processing method: Continuous reading/one-by-one reading
Obtaining/Displaying MICR data (same as Step 2)
Pasting electric endorsement (same as Step 3)
Obtaining the OCR-A/B font reading result (same as Step 6)
Step 8-4. Obtaining/Displaying the result of IQA process
BiGetIQAResult
Structure: MF_IQA_RESULT


Obtaining/displaying/storing read data (CALLBACK process)
Processing method: Continuous reading/one-by-one reading
Setting image quality (same as Step 5)
Obtaing/Displaying image data (same as Step 2)
Storing read data (same as Step 5)
Step 8-5. Obtaining/Displaying the result of Barcode decode
BiGetBarcodeData
Structure: MF_BARCODE
Process in the confirmation mode (same as Step 4)
Error handling process (same as Step 7)
(CALLBACK process)
Process to end use of the device (same as Step 1)

Step 8-1. Setting the IQA process

Set items of IQA analysis and operation when IQA error occurs. For details of setting items, see "MF_BARCODE structure" on page 3-11.

Setting MF_IQA

Set the members of MF_IQA structure with BiSCNMICRFunction*XXXX* before reading is processed. After setting, the obtained image data is analyzed and processed according to the analysis result.

- Calling initial values of the MF_IQA
- Changing values of members if necessary
- Setting values of the MF_IQA structure

1. Specify the memory address of the MF_IQA structure for the second parameter and MF_GET_IQA_DEFAULT for the third parameter of BiSCNMICRFunction*XXXX* to call initial values of the MF_IQA structure.

   nErr= BiSCNMICRFunctionContinuously (m_iHandle , *lpvStruct*, MF_GET_IQA_DEFAULT)

   Note:
   Use BiSCNMICRFunctionPostPrint when the process method is one-by-one reading.

2. Change the default values of members of the MF_IQA structure if necessary.
   For details of the members of MF_IQA structure and setting values, see "MF_IQA" on page 4-165.

   - Execution of the IQA process (bErrorSelect)

bErrorSelect	Description
MF_ERROR_SELECT_NODETECT (0)	Does not execute the IQA process.
MF_ERROR_SELECT_DETECT (1)	Executes the IQA process.

   - Ejection pocket of a document when IQA error occurs (bErrorEject)

bErrorEject	Description
MF_EJECT_MAIN_POCKET (0x22)	Ejects into the main pocket.
MF_EJECT_SUB_POCKET (0x24)	Ejects into the sub pocket.

   - Franking process when IQA error occurs (bStamp)

bStamp	Description
MF_STAMP_DISABLE (0)	Does not perform franking.
MF_STAMP_ENABLE (1)	Performs franking.

   - Continuation of reading process when IQA error occurs (bCancel;)

bCancel;	Description
MF_CANCEL_DISABLE (0)	Continues reading process.
MF_CANCEL_ENABLE (1)	Cancels reading process.

3. Specify the memory address of the MF_IQA structure for the second parameter and MF_SET_IQA_PARAM for the third parameter of the BiSCNMICRFunction*XXXX* to set the operation.

   nErr= BiSCNMICRFunctionContinuously (m_iHandle , *lpvStruct* , MF_SET_IQA_PARAM)

   Note:
   Use BiSCNMICRFunctionPostPrint when the process method is one-by-one reading.

Programming code

APIUsage.cpp CAPIUsage::ConfigureMultiple()

APIUsage.cpp CAPIUsage::ConfigureSingle()

```
m_tIqa.iSize = sizeof(MF_IQA);
m_tIqa.iVersion = MF_IQA_VERSION;

CheckResponse(BiSCNMICRFunctionContinuously(m_iHandle, &m_tIqa, MF_GET_IQA_DEFAULT));
SetIqaStruct();
CheckResponse(BiSCNMICRFunctionContinuously(m_iHandle, &m_tIqa, MF_SET_IQA_PARAM));
```

APIUsage.cpp CAPIUsage::SetIqaStruct()

```
m_tIqa.bErrorSelect = (BYTE)m_pProperties->GetValueDefFalse(IQA_DETECT);
m_tIqa.bErrorEject = ChangeEject(m_pProperties->GetValueDefFalse(IQA_EJECT));
m_tIqa.bStamp = (BYTE)m_pProperties->GetValueDefFalse(IQA_STAMP);
m_tIqa.bCancel = (BYTE)m_pProperties->GetValueDefFalse(IQA_CANCEL);

if(m_pProperties->GetValueDefFalse(IQA_GRAYSCALE))
{
        m_tIqa.bColorDepth = EPS_BI_SCN_8BIT;
        m_tIqa.bImageFormat = EPS_BI_SCN_JPEGNORMAL;
}
else
{
        m_tIqa.bColorDepth = EPS_BI_SCN_1BIT;
        m_tIqa.bImageFormat = EPS_BI_SCN_TIFF;
}

if(m_tIqa.bErrorSelect == MF_ERROR_SELECT_DETECT) {
        m_tIqa.bUndersize = MF_IQA_TEST_ENABLE;
        m_tIqa.bOversize = MF_IQA_TEST_ENABLE;
        m_tIqa.bMincompressed = MF_IQA_TEST_ENABLE;
        m_tIqa.bMaxcompressed = MF_IQA_TEST_ENABLE;
        m_tIqa.bFront_rear = MF_IQA_TEST_ENABLE;
        m_tIqa.bToolight = MF_IQA_TEST_ENABLE;
        m_tIqa.bToodark = MF_IQA_TEST_ENABLE;
        m_tIqa.bStreaks = MF_IQA_TEST_ENABLE;
        m_tIqa.bNoise = MF_IQA_TEST_ENABLE;
        m_tIqa.bFocus = MF_IQA_TEST_ENABLE;
        m_tIqa.bCorners = MF_IQA_TEST_ENABLE;
        m_tIqa.bEdges = MF_IQA_TEST_ENABLE;
        m_tIqa.bFraming = MF_IQA_TEST_ENABLE;
        m_tIqa.bSkew = MF_IQA_TEST_ENABLE;
        m_tIqa.bCarbon = MF_IQA_TEST_ENABLE;
        m_tIqa.bPiggyback = MF_IQA_TEST_ENABLE;
}
```

Step 8-2. Setting the barcode decode

Set the barcode decode process and behavior at barcode decode error.

Set the value to the member of MF_BARCODE structure with BiSCNMICRFunction*XXXX* before reading is processed. After setting, the setting of the barcode decode process can be specified.

- Calling initial values of the MF_BARCODE structure
- Changing values of members if necessary
- Setting values of the MF_BARCODE structure

1. Specify the memory address of the MF_BARCODE structure for the second parameter and MF_GET_BARCODE_FRONT_DEFAULT/ MF_GET_BARCODE_BACK_DEFAULT for the third parameter of BiSCNMICRFunction*XXXX* to call initial values of the MF_BARCODE structure.

   nErr= BiSCNMICRFunctionContinuously
   (m_iHandle , lpvStruct, MF_GET_BARCODE_FRONT_DEFAULT)

   nErr= BiSCNMICRFunctionContinuously
   (m_iHandle , lpvStruct, MF_GET_BARCODE_BACK_DEFAULT)

   Note:
   Use BiSCNMICRFunctionPostPrint when the process method is one-by-one reading.

2. Change the default values of members of the MF_BARCODE structure if necessary.
   For details of the members of MF_BARCODE structure and setting values, see "MF_BARCODE" on page 4-176.

3. Specify the memory address of the MF_BARCODE structure for the second parameter and MF_SET_BARCODE_FRONT_PARAM/MF_SET_BARCODE_BACK_PARAM for the third parameter of the BiSCNMICRFunction*XXXX* to set the operation.

   nErr= BiSCNMICRFunctionContinuously
   (m_iHandle , lpvStruct, MF_SET_BARCODE_FRONT_PARAM)

   nErr= BiSCNMICRFunctionContinuously
   (m_iHandle , lpvStruct, MF_SET_BARCODE_BACK_PARAM)

   Note:
   Use BiSCNMICRFunctionPostPrint when the process method is one-by-one reading.

Programming code

APIUsage.cpp CAPIUsage::SetBarcodeStruct()

```
void CAPIUsage::SetBarcodeStruct()
{
	m_tBarcodeFront.bErrorSelect = (BYTE)m_pProperties->GetValueDefFalse(BARCODE_FRONT_DETECT);
	m_tBarcodeFront.bErrorEject = ChangeEject(m_pProperties-
>GetValueDefFalse(BARCODE_FRONT_EJECT));
	m_tBarcodeFront.bStamp = (BYTE)m_pProperties->GetValueDefFalse(BARCODE_FRONT_STAMP);
	m_tBarcodeFront.bCancel = (BYTE)m_pProperties->GetValueDefFalse(BARCODE_FRONT_CANCEL);

	m_tBarcodeFront.stInfo[0].dwSymbolMask = (
MF_BARCODE_SYMBOL_CODABAR |
MF_BARCODE_SYMBOL_CODE128 |
MF_BARCODE_SYMBOL_CODE39 |
MF_BARCODE_SYMBOL_EAN_JAN |
MF_BARCODE_SYMBOL_ITF |
MF_BARCODE_SYMBOL_UPC_A |
MF_BARCODE_SYMBOL_UPC_E);

	m_tBarcodeBack.bErrorSelect = (BYTE)m_pProperties->GetValueDefFalse(BARCODE_BACK_DETECT);
	m_tBarcodeBack.bErrorEject = ChangeEject(m_pProperties-
>GetValueDefFalse(BARCODE_BACK_EJECT));
	m_tBarcodeBack.bStamp = (BYTE)m_pProperties->GetValueDefFalse(BARCODE_BACK_STAMP);
	m_tBarcodeBack.bCancel = (BYTE)m_pProperties->GetValueDefFalse(BARCODE_BACK_CANCEL);

	m_tBarcodeBack.stInfo[0].dwSymbolMask = (
MF_BARCODE_SYMBOL_CODABAR |
MF_BARCODE_SYMBOL_CODE128 |
MF_BARCODE_SYMBOL_CODE39 |
MF_BARCODE_SYMBOL_EAN_JAN |
MF_BARCODE_SYMBOL_ITF |
MF_BARCODE_SYMBOL_UPC_A |
MF_BARCODE_SYMBOL_UPC_E);
}
```

Step 8-3. Setting the Waterfall process

By calling BiSetWaterfallMode before reading is processed, the Waterfall process is performed. Specify Waterfall mode for the second parameter of BiSetWaterfallMode.

nErr= BiSetWaterfallMode (m_iHandle, WATERFALL_MODE_MAIN_*xxxx*)

WATERFALL_MODE_MAIN_xxxx	**Description**
WATERFALL_MODE_DISABLE (0)	Disables the Waterfall process.
WATERFALL_MODE_STANDARD (1)	Enables the Waterfall process. Ejects to the main pocket when starting the reading process. When the main pocket's near full is detected, the ejection pocket is switched to the sub pocket. When the sub pocket's near full is detected, the ejection pocket is switched to the main pocket.
WATERFALL_MODE_INHERIT_POCKET (2)	Enables the Waterfall process. Ejects to the ejection pocket of the previous reading process. (For example, the previous ejection pocket is the sub pocket, ejects to the sub pocket.) When a pocket near full is detected, the ejection pocket is switched to the other pocket. When a pocket near full has already been detected when starting the reading process, the ejection pocket is switched to the other pocket. The ejection pocket, however, is not switched when the other pocket's near full has been detected.

Note:

When the Waterfall process is enabled, the ejection pocket setting for MF_PROCESS structure is disabled.

Programming code

APIUsage.cpp CAPIUsage::Configure()

```
BYTE byMode = WATERFALL_MODE_DISABLE;
if(m_pProperties->GetValueDefFalse(WATERFALL_ENABLE)) {
        if(m_pProperties->GetValueDefFalse(WATERFALL_MODE) == WATERFALL_STANDARD) {
                byMode = WATERFALL_MODE_STANDARD;
        }
        else {
                byMode = WATERFALL_MODE_INHERIT_POCKET;
        }
}
BiSetWaterfallMode(m_iHandle, byMode);
```

Step 8-4. Obtaining/Displaying the result of IQA process

By calling BiGetIQAResult after obtaining/saving image data, the result of IQA process (MF_IQA_RESULT structure) can be obtained. For details of MF_IQA_RESULT structure, see "MF_IQA" on page 4-165.

Obtain the result of IQA process by specifying the transaction ID for the second parameter of BiGetIQAResult, the memory address of MF_IQA_RESULT structure for the third parameter.

nErr= BiGetIQAResult (m_iHandle, dwTransactionNumber, *ptIqaResult*)

Programming code

APIUsage.cpp CAPIUsage::ScanStatus

```
if(m_pProperties->GetValueDefFalse(IQA_DETECT)){
        MF_IQA_RESULT stIqaResult;
        stIqaResult.iSize = sizeof(MF_IQA_RESULT);
        stIqaResult.iVersion = MF_IQARESULT_VERSION;
        BiGetIqaResult(m_iHandle, dwTransactionNumber, &stIqaResult);
        pDlg->m_IqaResultDlg.SetIqaResult(&stIqaResult);
}
```

IqaResultDlg.cpp CIqaResultDlg::SetIqaResult(LPMF_IQA_RESULT lpResult)

```
// CIqaResultDlg message handlers
void CIqaResultDlg::SetIqaResult(LPMF_IQA_RESULT lpResult)
{
        DisplayResult(lpResult->stUnderSize.bResult, IDC_UNDER_SIZE_EDIT);
        DisplayResult(lpResult->stOverSize.bResult, IDC_OVER_SIZE_EDIT);
        DisplayResult(lpResult->stMinCompressedImageSize.bResult, IDC_MIN_COMP_EDIT);
        DisplayResult(lpResult->stMaxCompressedImageSize.bResult, IDC_MAX_COMP_EDIT);
        DisplayResult(lpResult->stFrontRearImageMismatch.bResult, IDC_MISMATCH_EDIT);
        DisplayResult(lpResult->stImageTooLight.bResult, IDC_TOO_LIGHT_EDIT);
        DisplayResult(lpResult->stImageTooDark.bResult, IDC_TOO_DARK_EDIT);
        DisplayResult(lpResult->stHorizontalStreaksPresent.bResult, IDC_STREAKS_EDIT);
        DisplayResult(lpResult->stExcessiveSpotNoise.bResult, IDC_NOISE_EDIT);
        DisplayResult(lpResult->stImageOutOfFocus.bResult, IDC_FOCUS_EDIT);
        DisplayResult(lpResult->stFoldedTornDocCorners.bResult, IDC_CORNERS_EDIT);
        DisplayResult(lpResult->stFoldedTornDocEdges.bResult, IDC_EDGES_EDIT);
        DisplayResult(lpResult->stDocFramingError.bResult, IDC_FRAM_EDIT);
        DisplayResult(lpResult->stExcessiveDocSkew.bResult, IDC_SKEW_EDIT);
        DisplayResult(lpResult->stCarbonStripDetection.bResult, IDC_STRIP_EDIT);
        DisplayResult(lpResult->stPiggyBack.bResult, IDC_PIGGYBACK_EDIT);
}
```

Step 8-5. Obtaining/Displaying the result of barcode decode (CALLBACK process)

Obtain and display the result of barcode decode with the CALLBACK function.

- Confirming data reception end
- Obtaining/Displaying the result of barcode decode

Confirming data reception end

Confirm the MF_DATARECEIVE_DONE status with the CALLBACK function to confirm data reception end.

Obtaining/Displaying the result of barcode decode

Follow the steps below to obtain/display the result of barcode decode.

1. Specify the side to obtain for the second parameter of BiSCNSelectScanFace.
   nErr= BiSCNSelectScanFace (m_iHandle , MF_SCAN_FACE_*xxxx*)

MF_SCAN_FACE_xxxx	Description
MF_SCAN_FACE_FRONT (0)	Selects the front side (default)
MF_SCAN_FACE_BACK (1)	Selects the back side

2. By specifying the transaction number for the second parameter and the memory address of the MF_BARCODE structure for the third parameter of BiGetBardodeData, the result of the barcode decode is obtained.

   nErr = BiGetBarcodeData (m_iHandle , dwTransactionNumber , ptBarcode)

3. The result of barcode decode returns to lpData of the MF_BARCODE structure.

4. The result of barcode decode is obtained in the CALLBACK function and displayed on the application.

Programming code

APIUsage.cpp CAPIUsage::ScanStatus()

```
DisplayBarcodeData(dwTransactionNumber);
```

APIUsage.cpp CAPIUsage::DisplayBarcodeData

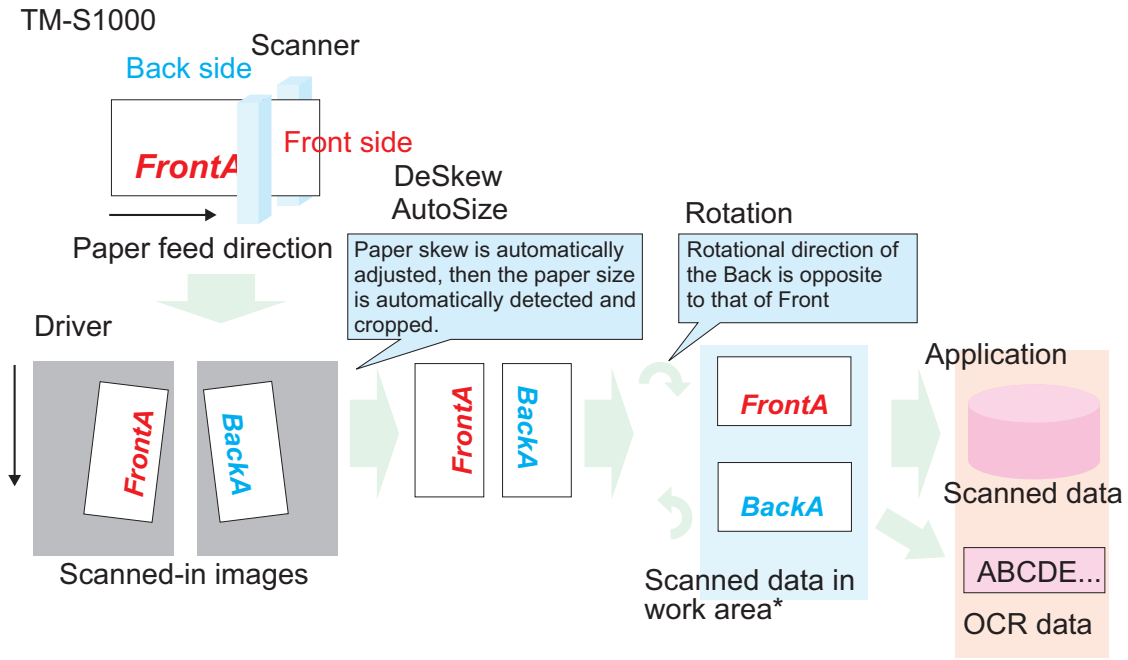
```
void CAPIUsage::DisplayBarcodeData(DWORD dwTransactionNumber)
{
	CTMS1000SampleDlg* pDlg = (CTMS1000SampleDlg*)theApp.m_pMainWnd;

	CString strBarcodeData;
	int iRet = SUCCESS;
	BARCODE_DATA *pBarcodeData = NULL;
	int nCount = 0;

	if(m_pProperties->GetValueDefFalse(BARCODE_FRONT_DETECT))
	{
		// select front side
		iRet = BiSCNSelectScanFace(m_iHandle, MF_SCAN_FACE_FRONT);
```

```
		if(iRet == SUCCESS)
		{
			iRet = BiGetBarcodeData(m_iHandle, dwTransactionNumber, &m_tBarcodeFront);
		}
		if((iRet == SUCCESS) && (m_tBarcodeFront.lpData != NULL))
		{
			// convert string
			pBarcodeData = (BARCODE_DATA*)m_tBarcodeFront.lpData;
			for(nCount = 0; nCount < m_tBarcodeFront.bDataCount; nCount++)
			{
				strBarcodeData.Append((LPSTR)pBarcodeData[nCount].pData,
							pBarcodeData[nCount].dwDataSize);
				strBarcodeData.Append(" ");
				::GlobalFree(pBarcodeData[nCount].pData);
			}
			strBarcodeData.Replace('\0', ' ');
			::GlobalFree(m_tBarcodeFront.lpData);
		}
		if(m_tBarcodeFront.iRet != SUCCESS)
		{
			strBarcodeData += GetResultString(m_tBarcodeFront.iRet);
		}
	}
	pDlg->SetBarcodeFrontString(strBarcodeData);
	strBarcodeData.Empty();

	if(m_pProperties->GetValueDefFalse(BARCODE_BACK_DETECT))
	{
		// select back side
		iRet = BiSCNSelectScanFace(m_iHandle, MF_SCAN_FACE_BACK);
		if(iRet == SUCCESS)
		{
			iRet = BiGetBarcodeData(m_iHandle, dwTransactionNumber, &m_tBarcodeBack);
		}
		if((iRet == SUCCESS) && (m_tBarcodeBack.lpData != NULL))
		{
			// convert string
			pBarcodeData = (BARCODE_DATA*)m_tBarcodeBack.lpData;
			for(nCount = 0; nCount < m_tBarcodeBack.bDataCount; nCount++)
			{
				strBarcodeData.Append((LPSTR)pBarcodeData[nCount].pData,
							pBarcodeData[nCount].dwDataSize);
				strBarcodeData.Append(" ");
				::GlobalFree(pBarcodeData[nCount].pData);
			}
			strBarcodeData.Replace('\0', ' ');
			::GlobalFree(m_tBarcodeBack.lpData);
		}
		if(m_tBarcodeBack.iRet != SUCCESS)
		{
			strBarcodeData += GetResultString(m_tBarcodeBack.iRet);
		}
	}
	pDlg->SetBarcodeBackString(strBarcodeData);
}
```

How to Use the Scanner Advanced Functions

When not using the scanner advanced functions

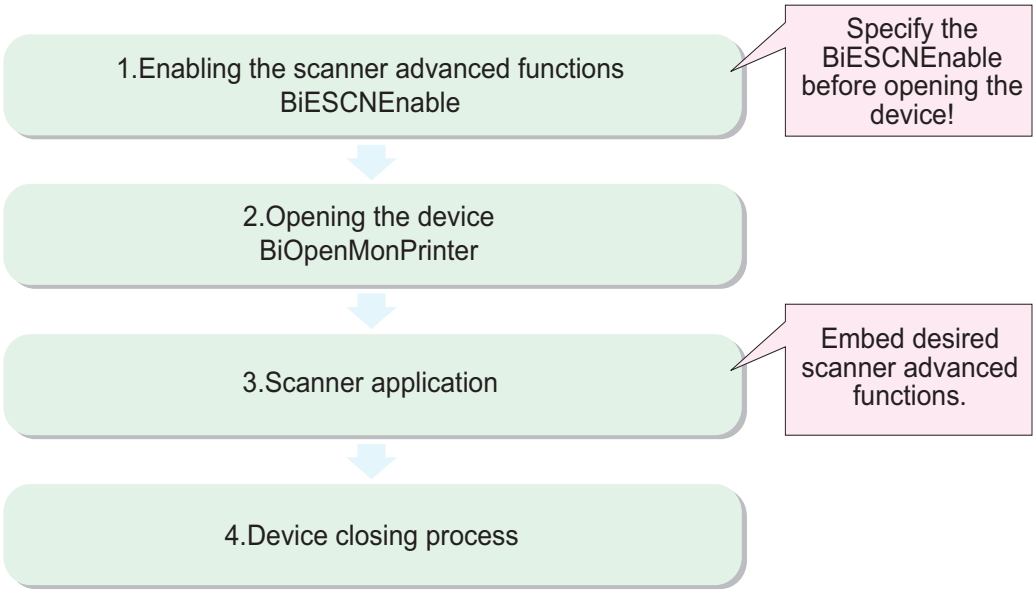
Scanned-in images are automatically edited and saved to the work area as shown below.



*OCR fonts in the scanned data saved in the work area can be read separately by specifying the area in which the fonts are included.
This can be carried out without using the scanner advanced functions.

Using the scanner advanced functions

Application process flow



Editing scanned-in images

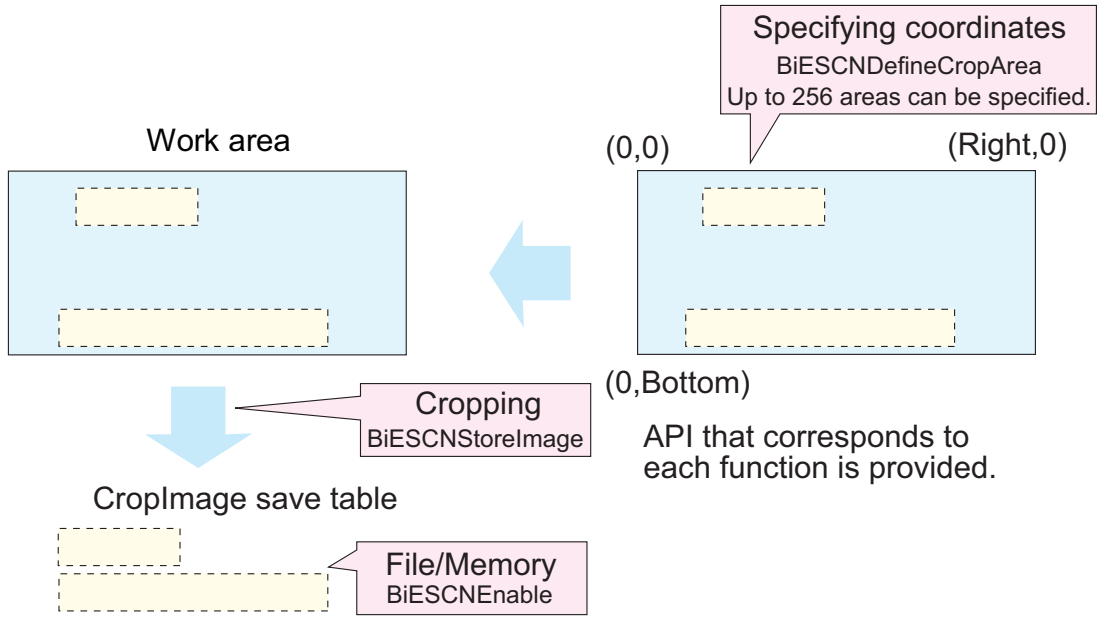
Images scanned by TM-S1000 can be edited using customer's application and saved to the work area.
The following settings are available with the scanner advanced functions.

Functional Classification	Item	API	Settings	When not using the advanced functions	Functions
Editing	Destination of the edited data	BiESCNEnable	Memory/File	-	Destination of edited scanned data can be specified in the work area. This must be specified in the beginning of the application whenever using the advanced functions.
	AutoSize	BiESCNSetAutoSize	Enable/ Disabled	Enable	Crops scanned-in image into a check sheet size by cropping out unnecessary part of the image.
	Skew adjustment	BiESCNSetDeSkew	Enable/ Disable/Enable when skew by specified or more degrees is detected	Skew by 1.5 degrees or more is adjusted	Skew of the scanned check sheet is detected and corrected.
	Rotation	BiESCNSetRotate	Enable/ Disabled	Enable	Front side is rotated clockwise while back side is rotated counterclockwise.

Note:
There is no difference in scanning speed between using or not using the scanner advanced functions.

Cropping

Areas to be cropped from the image data saved to the work area can be specified in advance using coordinates and can be saved.



Chapter 4
Reference

This chapter describes the TM-S1000 API and syntax.

Note
The data type is described in C++.

Device information

Device Status

Macro Definitions	ON/OFF	Value	Status	Response
ASB_NO_RESPONSE	ON	0x00000001	No Device response	Check that the device is powered on, and that the cable is connected. Check the specified device name, and also the computer port.
	OFF	0x00000000	Device Responds	-
ASB_OFF_LINE	ON	0x00000008	Off-line	Check the other Decice Status to eliminate them as a cause for the device being offline.
	OFF	0x00000000	On-line	-
ASB_COVER_OPEN	ON	0x00000020	Cover is open.	Close the cover.
	OFF	0x00000000	Cover is closed.	-
ASB_WAIT_PEPRT_EJECT	ON	0x00000100	There is a check paper in the transport path.	Remove the paper as described in the user's manual.
	OFF	0x00000000	-	-
ASB_MECHANICAL_ERR	ON	0x00000400	Recoverable Errors generated Refer to Technical Reference Guide	Check the other Device Status, eliminate the source of any errors, and then either power-on the scanner again or issue an error cancel (BiCancelError) command.
	OFF	0x00000000	No recoverable error.	-
ASB_UNRECOVER_ERR	ON	0x00002000	Unrecoverable Errors generated Refer to Technical Reference Guide	Immediately turn off the power to the device.
	OFF	0x00000000	No unrecoverable error.	-
ASB_PAPER_INTERMEDIATE	ON	0x00010000	The intermediate sensor senses no paper.	-
	OFF	0x00000000	The intermediate sensor senses that there is paper.	Remove the paper as described in the user's manual.

Macro Definitions	ON/OFF	Value	Status	Response
ASB_MAIN_NEAR_FULL	ON	0x00020000	The main pocket is not nearly full.	-
	OFF	0x00000000	The main pocket is nearly full.	Remove the paper from the main pocket.
ASB_EJECT_SENSOR_NO_PAPER	ON	0x00040000	The eject sensor senses no paper.	-
	OFF	0x00000000	The eject sensor senses that there is paper.	Remove the paper as described in the user's manual.
ASB_SUB_NEAR_FULL	ON	0x00080000	The sub pocket is not nearly full.	-
	OFF	0x00000000	The sub pocket is nearly full.	Remove the paper from the sub-pocket.
ASB_SLIP_PAPER_SIZE	ON	0x00200000	No check paper at the paper length sensor.	-
	OFF	0x00000000	Check paper at the paper length sensor.	Remove the paper as described in the user's manual.
ASB_ASF_PAPER	ON	0x00400000	There is no paper in the ASF.	-
	OFF	0x00000000	There is paper in the ASF.	-
ASB_STAMP_EXIST	ON	0x02000000	The franking cartridge is not installed.	When ASB_STAMP_EXIST = NO and the Franker is enable, the application needs to prompt a warning.
	OFF	0x00000000	The franking cartridge is installed.	-
ASB_WAIT_INSERT	ON	0x20000000	Waiting for paper.	-
	OFF	0x00000000	-	-
ASB_FRANKING_SENSOR	ON	0x40000000	The franking sensor senses no paper.	-
	OFF	0x00000000	The franking sensor senses there is paper.	Remove the paper as described in the user's manual.

Maintenance Counter

Counter Number (read no)	Resetability	Counter (readcounter)	(Unit)
60 (3CH) *1	Resetable	Magnetic text read count	(No of times)
61 (3DH) *1	Resetable	Check paper image read count	(No of times)
70 (46H)	Resetable	Product operating time	(Hours)
80 (50H)	Resetable	Hopper open/close count	(No of times)
81 (51H)	Resetable	Franking count	(No of times)
82 (52H)	Resetable	Pocket switching count	(No of times)
188 (BCH) *2	Cumulative	Magnetic text read count	(No of times)
189 (BDH) *2	Cumulative	Check paper image read count	(No of times)
198 (C6H)	Cumulative	Product operating time	(Hours)
208 (D0H)	Cumulative	Hopper open/close count	(No of times)
209 (D1H)	Cumulative	Franking count	(No of times)
210 (D2H)	Cumulative	Pocket switching count	(No of times)

*1 The values set to 60 (3CH) and 61 (3DH) are the same. If you reset one of them, the other one is also reset.
*2 The values set to 188 (BCH) and 189 (BDH) are the same.

Type ID

Bit	ON/OFF	Value	Function
0	-	0x00	-
1	-	0x00	-
2	-	0x00	-
3	ON	0x08	MICR reader is installed. (Fixed to 1)
4	-	0x00	Fixed to 0
5	-	0x00	-
6	-	0x00	-
7	-	0x00	Fixed to 0

Device ID

Device ID (Set to *prnID*)	Type of Device ID	Data to obtain
1	Product ID	1-byte data. Product ID is 111(6FH).
2	Type ID	1-byte data. See Table (A) for the data to obtain.
33	Type information (1)	2-byte data. See Table (B) for the data to obtain.
65	Version of firmware	String data. The obtained data varies depending on the firmware version. Example: "1.00 ESC/POS"
66	Manufacture name	String data. Default is "EPSON". Example: "EPSON"
67	Product name	String data. The obtained data varies depending on the prodct. Example: "TM-S1000"
68	Serial number	String data. The obtained data varies depending on the device. Example: "DEKK000015"
112	Type information (2)	1-byte data. See Table (C) for data to obtain.

(A) Type ID (Device ID = 2)

Bit	ON/OFF	Value	Function
0	-	0x00	-
1	-	0x00	-
2	-	0x00	-
3	ON	0x08	MICR reader is installed. (Fixed to 1)
4	-	0x00	-
5	-	0x00	-
6	-	0x00	-
7	-	0x00	-

(B) Type information (Device ID = 33)

First byte

Bit	ON/OFF	Value	Function
0	-	0x00	-
1	-	0x00	-
2	-	0x00	-
3	ON	0x08	MICR reader is installed. (Fixed to 1)
4	ON	0x10	Image reader is installed. (Fixed to 1)
5	-	0x00	-
6	-	0x40	Fixed to 1
7	-	0x00	Fixed to 0

Second byte

Bit	ON/OFF	Value	Function
0	-	0x00	-
1	ON	0x02	Grayscale reading is supported. (Fixed to 1)
2	ON	0x04	Image reader is installed. (Fixed to 1)
3	ON	0x08	ASF is installed. (Fixed to 1)
4	-	0x00	-
5	-	0x00	-
6	-	0x40	Fixed to 1
7	-	0x00	Fixed to 0

(C) Type information (Device ID = 112)

Bit	ON/OFF	Value	Function
0	-	0x00	-
1	Model (Refer to "Model" on page 4-5)		
2			
3	-	0x00	-
4	ON	0x10	Waterfall is supported.
	OFF	0x00	Waterfall is unsupported.
5	OFF	0x00	2 pockets (Fixed to 0)
6	-	0x40	Fixed to 1
7	-	0x00	-

Model

Model	1Bit	2Bit
30 dpm	OFF (0x00)	OFF (0x00)
60 dpm	ON (0x02)	OFF (0x00)
90 dpm	ON (0x02)	ON (0x04)

Offline Code (BiGetOfflineCode)

First Byte

Bit	ON/OFF	Value	Status
0	ON	0x02	CPU execution error generated
	OFF	0x00	CPU execution error not generated
1	ON	0x04	ROM error generated
	OFF	0x00	ROM error not generated
2	-	0x00	fixed to 0
3	-	0x00	fixed to 0
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Second Byte

Bit	ON/OFF	Value	Status
0	ON	0x01	High voltage error generated
	OFF	0x00	High voltage error not generated
1	ON	0x02	Low voltage error generated
	OFF	0x00	Low voltage error not generated
2	-	0x00	fixed to 0
3	-	0x00	fixed to 0
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Third Byte

Bit	ON/OFF	Value	Status
0	-	0x00	fixed to 0
1	-	0x00	fixed to 0
2	-	0x00	fixed to 0
3	-	0x00	fixed to 0
4	ON	0x04	CIS error generated
	OFF	0x00	CIS error not generated
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Fourth Byte

Bit	ON/OFF	Value	Status
0	-	0x00	fixed to 0
1	-	0x00	fixed to 0
2	-	0x00	fixed to 0
3	-	0x00	fixed to 0
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Fifth Byte

Bit	ON/OFF	Value	Status
0	-	0x00	fixed to 0
1	-	0x00	fixed to 0
2	-	0x00	fixed to 0
3	-	0x00	fixed to 0
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Offline Code (BiGetOfflineCodeByIndex)

First Byte (Recoverable Errors)

Bit	ON/OFF	Value	Status
0	-	0x00	fixed to 0
1	ON	0x02	CPU execution error generated
	OFF	0x00	CPU execution error not generated
2	ON	0x04	ROM error generated
	OFF	0x00	ROM error not generated
3	-	0x00	fixed to 0
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Second Byte (Unrecoverable Errors)

Bit	ON/OFF	Value	Status
0	ON	0x01	High voltage error generated
	OFF	0x00	High voltage error not generated
1	ON	0x02	Low voltage error generated
	OFF	0x00	Low voltage error not generated
2	ON	0x04	CIS error generated
	OFF	0x00	CIS error not generated
3	-	0x00	fixed to 0
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 1

Third Byte (Recoverable Errors: mechanism position error)

Bit	ON/OFF	Value	Status
0	ON	0x01	Hopper position error generated
	OFF	0x00	Hopper position error not generate
1	ON	0x02	Stamp position error generated
	OFF	0x00	Stamp position error not generated
2	ON	0x04	Switching plate position error generated
	OFF	0x00	Switching plate position error not generated
3	-	0x00	fixed to 0
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Fourth Byte (Recoverable Errors: paper jam error)

Bit	ON/OFF	Value	Status
0	ON	0x01	Cut paper present error generated
	OFF	0x00	Cut paper present error not generated
1	ON	0x02	Cut-sheet ejection error generated
	OFF	0x00	Cut-sheet ejection error not generated
2	ON	0x04	Cut paper length error generated
	OFF	0x00	Cut paper length error not generated
3	ON	0x08	Cut paper transport error generated
	OFF	0x00	Cut paper transport error not generated
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 1
7	-	0x00	fixed to 0

Fifth Byte (Recoverable Errors: continuous read error that detection setting is available)

Bit	ON/OFF	Value	Status
0	ON	0x01	Cut paper double feed error generated
	OFF	0x00	Cut paper double feed error not generated
1	ON	0x02	Cut paper insertion direction error generated
	OFF	0x00	Cut paper insertion direction error not generated
2	ON	0x04	External noise error generated
	OFF	0x00	External noise error not generated
3	ON	0x08	Read error generated as a result of command indication
	OFF	0x00	Read error not generated as a result of command indication
4	-	0x00	fixed to 0
5	-	0x00	fixed to 0
6	-	0x40	fixed to 0
7	-	0x00	fixed to 0

MICR Status

Bit	ON/OFF	Value	Status
0	-	0x00	fixed to 0
1	-	0x02	fixed to 1
2	ON	0x04	MICR function non-selection
	OFF	0x00	MICR function selection
3	ON	0x08	Waiting for check sheet/cleaning sheet insertion
	OFF	0x00	-
4	-	0x10	fixed to 1
5	ON	0x20	TOF sensor senses no paper
	OFF	0x00	TOF sensor senses that there is paper
6	ON	0x40	BOF sensor senses no paper
	OFF	0x00	BOF sensor senses that there is paper
7	-	0x00	fixed to 0

TM-S1000 API Error Handling

The following table shows how TM-S1000 API handles each of the error return values.

Constant	Description	Response
SUCCESS	Success	-
ERR_TYPE	nType parameter error	A constant value that is not defined is used in the header. Use a constant value that is defined.
ERR_OPENED	The specified device has already been opened	The port is already used by the other application. Close the application.
ERR_NO_PRINTER	The specified device driver does not exist	The power of the device is off, or the host is not connected. Check the device status.
ERR_NO_TARGET	An unsupported device was specified (The device's power is not On or the cable connections are faulty, etc.)	The current USB driver is not supported. Install the correct driver.
ERR_NO_MEMORY	Memory is insufficient	There is insufficient memory for running the driver. Close other applications or add more memory.
ERR_HANDLE	The handle value that specifies the device is incorrect	The handle value is incorrect. Check if the same handle value is used as the one when BiOpenMonPrinter is correctly finished.
ERR_TIMEOUT	A time out error occurred	API is not finished correctly. Check the device status.
ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)	API is not finished correctly. Check the device status.
ERR_PARAM	Parameter error	This is a parameter error. Check if the set value is correct.
ERR_NOT_SUPPORT	Unsupported	This API function is not available. When using the scanner advanced function API, confirm that BiESCNEnable is called at the beginning.
ERR_OFFLINE	It was opened in the offline state, so it cannot be used until the online state is recovered.	The device has gone offline. Check the device status.
ERR_NOT_EPSON	Cannot be used (device not EPSON)	The current device is not EPSON product. Use an EPSON's device.
ERR_WITHOUT_CB	BiSCNMICRFunctionPostPrint/ BiSCNMICRFunctionContinuously has not been run	• The callback destination is not registered on the driver. Call either BiSCNMICRSetStatusBackFunction or BiSCNMICRSetStatusBackWnd • The scanner is not in operation. Call it while the scanner is scanning.
ERR_BUFFER_OVER_FLOW	Buffer overflow error	The memory is insufficient for the required task. Add more memory.
ERR_ENABLE	Cannot be used because BiOpenMonPrinter is called	BiOpenMonPrinter is already called. Execute BiCloseMonPrinter, and then recall BiOpenMonPrinter.
ERR_DISK_FULL	There is insufficient free space on the disk	Disk capacity is insufficient. Review the operating environment.
ERR_NO_IMAGE	The image data does not exist	No image files are found. Check if the image file exists in the specified destination.

Constant	Description	Response
ERR_CROPAREAID	The specified Crop Area does not exist	DefineCropArea is not configured. Set DefineCropArea, and then recall it.
ERR_EXIST	The specified data has already been saved	The parameter has already been registered. Change the parameter or call BiESCNClearImage.
ERR_NOT_FOUND	No data error	• No data exist in the corresponding transaction number. Confirm that the transaction number is correct. • It is not registered yet. Check the parameter. • No corresponding module is found. Re-install the driver.
ERR_IMAGE_FILEOPEN	Open failure	Specified image file cannot be opened. Check if the other application is using it.
ERR_IMAGE_UNKNOWNFORMAT	Format injustice	File format is incorrect or not supported. Check if the image can be opened with the other application.
ERR_IMAGE_FAILED	Image data creation failed	Saving image file failed. Review the operating environment.
ERR_IMAGE_FILEREAD	Read of the image data file failed	Reading image file failed. Check if the file exists in the specified destination.
ERR_PAPERINSERT_TIMEOUT	Paper insertion time exceeded	Inserting the check sheet failed. Check if the check paper is correctly placed on ASF.
ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed	The other API is in use. Retry after the other API is finished.
ERR_RESET	Cannot be used because the device is being reset	The device is being reset. Retry after the reset is finished.
ERR_ABORT	Canceled by BiSCNMICRCancelFunction	Reading operation is canceled. Determine whether to execute the operation again or not.
	Near full is detected	Notified when MF_PROCESS.bNearFullSelect is set to other than MF_NEARFULL_PERMIT and Near full is detected during scanning.
ERR_MICR	Printer failed in MICR reading	MICR data is not read. Read it again.
ERR_SCAN	Printer failed in image scanning	Image data is not read. Read it again.
ERR_NOT_EXEC	Process not being executed	Reading operation is not executed. Read it again.
ERR_SIZE	Size excess error	BiSetPrintSize is not called, or the value is not valid. Execute it again after calling BiSetPrintSize.
ERR_PAPER_PILED	Paper pilling error	Double feed error is detected. Determine whether to execute the reading operation again or not.
ERR_PAPER_JAM	Paper jam has occurred	Paper jam error occurs. Remove the jammed paper and call BiCancelError.
ERR_COVER_OPEN	Cover open error	Path cover is opened. Close the cover.
ERR_MICR_NODATA	MICR data is not existing	MICR data does not exist. Determine whether to execute the reading operation again or not.

Constant	Description	Response
ERR_MICR_BADDATA	MICR data is not able to recognize	MICR data is incorrect. Determine whether to execute the reading operation again or not.
ERR_MICR_PARSE	MICR data can not be parsed	Parsing operation of MICR data failed. Determine whether to execute the reading operation again or not.
ERR_MICR_NOISE	Noise error has occurred during MICR reading	External noise error is detected. Review the installation site, and determine whether to execute the reading operation again or not.
ERR_PAPER_EXIST	API can not be execute because there is a paper on the path	Paper exists on the paper feed path. Remove the paper.
ERR_PAPER_INSERT	Paper insertion error	Paper insertion error is detected. Check if the insertion direction is correct, and judge whether to execute the reading operation again or not.
ERR_LESS_CHECKS	The number of documents specified by BiSetNumberOfDocuments cannot be read.	Change the specified number of documents, or increase the number of documents to be read.
ERR_SCAN_IQA	Error has detected in the IQA test.	Call BiGetIQAResult to confirm the test result, and determine if reading should be tried again.
ERR_BARCODE_NODATA	Barcode cannot be detected.	• Check if the specifications of front/back side are correct. • Check if the specified barcode type and the decode direction are correct. • Read image data may be low in quality. Read it again then.

BiOpenMonPrinter

This opens the device for which the status is being monitored. Values acquired by this API are used as nHandle for other APIs.

Syntax

int **BiOpenMonPrinter** *(int nType, LPSTR pName)*

Argument

nType: This specifies the type of name specified in *pName*. One of the following two types is specified. This is an INT type.

Constant	Value	Description
TYPE_PORT	1	Specify the port name in pName.
TYPE_PRINTER	2	Specify the device name in pName.

pName: This specifies the device that is opened. The specification is as follows, depending on the nType value. If the *nType* is TYPE_PORT, it specifies the port name to which the device is connected.
(Example: "USB2"...)
If the *nType* is TYPE_PRINTER, the device name is specified.
(Example: "TM-S1000U", ...)
This is an LPSTR type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-10	ERR_TYPE	nType parameter error
-20	ERR_OPENED	The specified device has already been opened
-30	ERR_NO_PRINTER	The specified device driver does not exist
-40	ERR_NO_TARGET	An unsupported device was specified (The device's power is not On or the cable connections are faulty, etc.)
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-120	ERR_NOT_EPSON	Cannot be used (device not EPSON)

Explanation

Before using an API function other than this function, it is necessary that this function be executed first.
The handle value obtained is valid only in the same thread.

Note

For Windows 2000, Windows XP, Windows Vista or newer Windows versions., if the common memory has failed to be secured because of the user authority, ERR_NO_PRINTER or ERR_NO_MEMORY is returned.

BiSetMonInterval

API that is compatible with the TM-J9000 API. No operation is possible with the TM-S1000 API.

Syntax

int **BiSetMonInterval** (*int nHandle, WORD wNoPrnInterval, WORD wPrnInterval*)

Argument

nHandle: Specifies the handle. This is an INT type.

wNoPrnInterval: Not used

wPrnInterval: Spooler status monitoring interval is set in units of msec.
This is a WORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

BiGetStatus

Acquires the current device status. The device status is set to the lpStatus.

Syntax

int **BiGetStatus** *(int nHandle, LPDWORD lpStatus)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpStatus: The current status of the device is set. This is an LPDWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error

Explanation

Refer to "Device Status" on page 4-1, for the device statuses that you can acquire.

BiSetStatusBackFunction

This registers the called callback function on the occasion of device status is changed.

Note

This is unavailable when the development environment is VB.

Syntax

int **BiSetStatusBackFunction** *(int nHandle, int (CALLBACK EXPORT *pStatusCB) (DWORD dwStatus))*

Argument

nHandle: Specifies the handle. This is an INT type.

*(CALLBACK EXPORT *pStatusCB) (DWORD dwStatus):*
Sets the callback function's address. This is an INT type.

Callback function parameters

dwStatus: The current status of the device is set. This is a DWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

When the device status has changed, the 4 bytes of the Device status that are cast as DWORD are saved to dwStatus. For details on the Device status, refer to "Device Status" on page 4-1.

BiSetStatusBackFunctionEx

This registers the called callback function on the occasion of device status is changed. Identifies the port originating the CALLBACK, in addition to the functions of BiSetStatusBackFunction.

Note

This is unavailable when the development environment is VB.

Syntax

int **BiSetStatusBackFunctionEx** (*int nHandle,*
*int (CALLBACK EXPORT *pStatusCB)(DWORD dwStatus,*
LPSTR lpcPortName)

Argument

nHandle: Specifies the handle. This is an INT type.

*int (CALLBACK EXPORT *pStatusCB)(DWORD dwStatus, LPSTR lpcPortName):*
Sets the callback function's address. This is an INT type.

Callback function parameters

dwStatus: The Device status held by the TM-S1000 API is set. This is a DWORD type.

lpcPortName: Returns the name of the CALLBACK printer port. This is an LPSTR type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

When the device status has changed, the 4 bytes of the Device status that are cast as DWORD are saved to dwStatus. For details on the Device status, refer to "Device Status" on page 4-1. Also, such that CALLBACK can be confirmed from any port, the port names are set in 1pcPortName

BiSetStatusBackWnd

Upon a status notification, this records the handle of the button that sends a button click event and the address of the memory that sets the status information.

Syntax

int **BiSetStatusBackWnd**(*int nHandle, long hWnd, LPDWORD lpStatus)*

Argument

nHandle: Specifies the handle. This is an INT type.

hWnd: This specifies the handle of the button that sends a button click event upon at status notification.

lpStatus: Sets the status value in lpStatus and issues an event.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

When there has been a change in the status of the device, posts notification of a click event being sent for the specified button. For details on the status values, see "Device Status" on page 4-1.

BiCancelStatusBack

This cancels Automatic Status Back registered by BiSetStatusBackFunction, BiSetStatusBackFunctionEx or BiSetStatusBackWnd.

Syntax

int **BiCancelStatusBack** *(int nHandle)*

Argument

nHandle: Specifies the handle. This is an INT type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect

Explanation

The cancel request is ignored and SUCCESS return if a status notification request has not been registered.

BiResetPrinter

Resets the device.

Syntax

int **BiResetPrinter** *(int nHandle)*

Argument

nHandle: Specifies the handle. This is an INT type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

BiGetCounter

Acquires the maintenance counter value.

Syntax

int **BiGetCounter** (*int nHandle, WORD readno, LPDWORD readcounter*)

Argument

nHandle: Specifies the handle. This is an INT type.

readno: Specifies the number of the acquired maintenance counter. This is a WORD type.

readcounter: Returns the maintenance counter. This is an LPDWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	It was opened in the offline state, so it cannot be used until the online state is recovered.
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Refer to "Maintenance Counter" on page 4-3 regarding the device counter and the acquired maintenance counters.
The maintenance counters include those that can be reset by the user and those that are not reset and count.

BiResetCounter

Resets the maintenance counter.

Syntax

int **BiResetCounter** (*int nHandle, WORD writeno*)

Argument

nHandle: Specifies the handle. This is an INT type.

writeno: Specifies the number of the reset maintenance counter. This is a WORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	It was opened in the offline state, so it cannot be used until the online state is recovered.
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The maintenance counters include those that can be reset by the user and those that are not reset and count. Refer to "Maintenance Counter" on page 4-3.

Note

In order for this API to be compatible with the TM-J9000 API, the magnetic text read count and check sheet image read count are defined separately, but the maintenance counters are stored as the read count. As a result, resetting either the magnetic text read count or the check sheet read count resets both counts.

BiCancelError

Restores recoverable device errors.

Syntax

int **BiCancelError***(int nHandle)*

Argument

nHandle: Specifies the handle. This is an INT type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)

BiGetType

Acquires the device type ID. The TM-S1000 is not used as it acquires meaningless information. This API is compatible with the TM-J9000.

Syntax

int **BiGetType** (*int nHandle, LPBYTE typeID, LPBYTE font, LPBYTE exrom, LPBYTE special*)

Argument

nHandle: Specifies the handle. This is an INT type.

typeID: Returns the device type ID. This is an LPBYTE type.

font: Not used

exrom: Not used

special: Not used

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	It was opened in the offline state, so it cannot be used until the online state is recovered.
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Refer to "Type ID" on page 4-3.

BiGetOfflineCode

Acquires the cause of the device to go offline. This API is compatible with the TM-J9000.

Syntax

int **BiGetOfflineCode** (*int nHandle, LPBYTE lpOfflinecode*)

Argument

nHandle: Specifies the handle. This is an INT type.

offlinecode: Returns the 5-byte value indicating the reason for the device going offline. This is an LPBYTE type.

> **Note**
> *When the cause of the device going offline is acquired while the device is online, zero will be written to the leading byte of lpOfflinecode.*

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Refer to "Offline Code (BiGetOfflineCode)" on page 4-6.

BiGetOfflineCodeByIndex

Acquires the cause of the device to go offline from the leading byte.

Syntax

int **BiGetOfflineCodeByIndex** (*int nHandle, int nIndex, LPBYTE lpOfflinecode*)

Argument

nHandle: Specifies the handle. This is an INT type.

nIndex: Specifies bytes (1 to 5) of the acquired cause of the device going offline. This is an INT type.

offlinecode: Returns the 5-byte value indicating the reason for the device going offline. This is an LPBYTE type.

Note

When the cause of the device going offline is acquired while the device is online, zero will be written to the leading byte of lpOfflinecode.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Refer to "Offline Code (BiGetOfflineCodeByIndex)" on page 4-8.

BiMICRSelectDataHandling

Specifies an operation when an error occurs during reading check paper. This API is compatible with the TM-J9000.

Syntax

int **BiMICRSelectDataHandling***(int nHandle, BYTE charSelect, BYTE detailSelect, BYTE errorSelect)*

Argument

nHandle: Specifies the handle. This is an INT type.

charSelect: Specifies how to handle the characters unable to be analyzed. This is a BYTE type.

Value	Description
0	Terminates analyzing characters when a character that is unable to be analyzed has been detected, and does not save the read data.
1	Replaces the characters that are unable to be analyzed with '?,' and continues to analyze characters.

detailSelect: Not used.

errorSelect: Specifies whether or not to terminate reading when an error has occurred. When the reading process ends normally or when an error occurs while the reading result is being saved, ignores errorSelect and continues reading. The values below can be specified solely or two among them together.

Constant	Value	Description
ES_STOP_ALL	0	Terminates reading when an error has occurred.
ES_CONTINUE_ALL	1	Continues the reading process when an error occurs if continuing is possible. The 4 errors where continuing is possible are double feed, magnetic waveform detection error,unrecognized character detection error, and noise error.
ES_CONTINUE_DOUBLEFEED	2	Continues reading even after a double feeding error has occurred.
ES_CONTINUE_NODATA	4	Continues reading even after a magnetic waveform detection error has occurred.
ES_CONTINUE_BADDATA	8	Continues reading even after an unanalyzable character detection error has occurred.
ES_CONTINUE_NOISE	16	Continues reading even after a noise error has occurred.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Note

For compatiblility with the TM-J9000 driver, when MF_PROCESS structure is not set and the initial value defined with the MF_PROCESS structure and the action defined with *errorSelect* are noneqivalent, the priority is given to the action defined with *errorSelect*.

BiMICRGetStatus

Acquires the MICR status. This API is compatible with the TM-J9000.

Syntax

int **BiMICRGetStatus** (*int nHandle, LPBYTE pStatus*)

Argument

nHandle: Specifies the handle. This is an INT type.

pStatus: Specifies the memory address that acquires the MICR status. This is an LPBYTE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	It was opened in the offline state, so it cannot be used until the online state is recovered.
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Refer to "MICR Status" on page 4-10 regarding the acquired MICR status.

BiMICRCleaning

Cleans the MICR mechanism.

Syntax

int **BiMICRCleaning** *(int nHandle)*

Argument

nHandle: Specifies the handle. This is an INT type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-110	ERR_OFFLINE	It was opened in the offline state, so it cannot be used until the online state is recovered.

Explanation

If this function is called, the mechanism waits for the cleaning sheet to be inserted. Insert the cleaning sheet and carry out cleaning of the mechanism.

BiSCNSetImageQuality

Sets the image scanning quality. 256 gray scale (8 bit) is used for setting the sharpness, and Black and White (1 bit) is used for setting the Threshold.

Syntax

int **BiSCNSetImageQuality**(*int nHandle, BYTE bColorDepth, char bThreshold, BYTE bColor,BYTE bExOption)*

Argument

nHandle: Specifies the handle. This is an INT type.

bColorDepth: Specifies the tonal gradation (the number of bits used for 1 pixel). The valid specification values are 1 or 8. The default value is EPS_BI_SCN_1BIT. This is a BYTE type.

Constant	Value	Description
EPS_BI_SCN_1BIT	1	Black and White (1 bit)
EPS_BI_SCN_8BIT	8	256 gray scale (8 bit)

bThreshold: Brightness threshold value (-128 ~ 127) for Black and White. Used when bColorDepth=EPS_BI_SCN_1BIT, and bExOption=EPS_BI_SCN_MANUAL. The value -128 to 127 corresponds to 0 to 255 of the brightness. When "0" is specified, the intermediate brightness "128" is applied. This is a char type.

bColor: Specifies the color. Valid specification values are as shown below. This is a BYTE type. However, in the current version, this value is fixed at EPS_BI_SCN_MONOCHROME, and any other value that is specified is regarded as invalid.

Constant	Value	Description
EPS_BI_SCN_MONOCHROME	48	Monochrome
EPS_BI_SCN_COLOR	49	TM-S1000 API does not support this.

bExOption: Specifies density adjustment types. Valid specification values are as shown below. This is a BYTE type.

Constant	Value	Description
EPS_BI_SCN_MANUAL	49	Applies the value of bThreshold for Black and White.
EPS_BI_SCN_SHARP	50	Sharpening
EPS_BI_SCN_SHARP_CUSTOM	51	Sharpening (for compatibility with TM-J9000 API. The setting is the same as EPS_BI_SCN_SHARP)
EPS_BI_SCN_SHARP_CUSTOM2	52	Sharpening (for compatibility with TM-J9000 API. The setting is the same as EPS_BI_SCN_SHARP)
EPS_BI_SCN_SHARP_CUSTOM3	53	Edge-preserving smoothing Recommended when JPEG 2000 is used with your application.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Refer to the following for details on this API

bColorDepth	bExOption	Scan result
EPS_BI_SCN_1BIT	EPS_BI_SCN_MANUAL	A value set to bThreshold is applied
	EPS_BI_SCN_SHARP	bThreshold is automatically set and the set value is applied
EPS_BI_SCN_8BIT	EPS_BI_SCN_MANUAL	No special handling is applied.
	EPS_BI_SCN_SHARP	Sharpening

This function is valid for the scanner unit currently selected.

- ❏ If bColorDepth is set to EPS_BI_SCN_8BITS(8), the parameter bThreshold is ignored.
- ❏ In the case of Executing BiSCNMICRFunction,The set reading quality is applied beginning with the next image data read. (Reading quality is not applied to image data that has already been read.)

BiSCNSetImageFormat

Selects the format of the scanning image data. The selected format is enabled until BiCloseMonPrinter is executed.

Syntax

int **BiSCNSetImageFormat***(int nHandle, BYTE bFormat)*

Argument

nHandle: Specifies the handle. This is an INT type.

bFormat: Specifies the format of image data notified. The valid specification values are as shown below. The default value is EPS_BI_SCN_TIFF(1). This is a BYTE type.

Constant	Value	Description	8 Bit	1 Bit
EPS_BI_SCN_TIFF	1	TIFF format CCITT (Group 4) compressed data	-	✔
EPS_BI_SCN_RASTER	2	Raster format uncompressed data	✔	-
EPS_BI_SCN_BITMAP	3	Bitmap format uncompressed data	✔	✔
EPS_BI_SCN_TIFF256	4	TIFF format uncompressed data	✔	-
EPS_BI_SCN_JPEGHIGH	5	JPEG format high compression (size priority) data	✔	-
EPS_BI_SCN_JPEGNORMAL	6	JPEG format normal compression data	✔	-
EPS_BI_SCN_JPEGLOW	7	JPEG format low compression (quality priority) data	✔	-
EPS_BI_SCN_JTIFF	8	TIFF format JPEG compressed data	✔	-

Note

To save as an image file of ANSI X9.100-181-2007 standard, it must be a TIFF format with CCITT(Group 4) compression data (bFormat = EPS_BI_SCN_TIFF), black and white (bColorDepth of BiSCNSetImageQuality = EPS_BI_SCN_1BIT), and whose resolution is 200 dpi (sResolution of MF_SCAN = MF_SCAN_DPI_200).

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

This function is valid for the scanner unit currently selected.

In the case of Executing BiSCNMICRFunction,The set reading quality is applied beginning with the next image data read. (Reading quality is not applied to image data that has already been read.)

When using BiSCNMICRFunctionPostPrint or BiSCNMICRFunctionContinuously, the image data format setting made by this API can be applied even after acquiring an image using BiGetScanImage in MF_DATA_RECEIVE_DONE callback. To do that, call BiGetScanImage again after setting the image data format using this API.

BiSCNSetScanArea

Configures the reading area of the image data. This API is compatible with the TM-J9000.

Syntax

int **BiSCNSetScanArea** (*int nHandle, BYTE bStartX, BYTE bStartY, BYTE bEndX, BYTE bEndY*)

Argument

nHandle: Specifies the handle. This is an INT type.

bStartX: Specifies the start X coordinate (0 to 98) of the read area in units of mm.
This is a BYTE type. The initial value is 0.

bStartY: Specifies the start Y coordinate (0 to 228) of the read area in units of mm.
This is a BYTE type. The initial value is 0.

bEndX: Specifies the end X coordinate (0 to 100) of the read area in units of mm.
This is a BYTE type. The initial value is 70.
When 0 is specified, the maximum value supported by the device is specified.

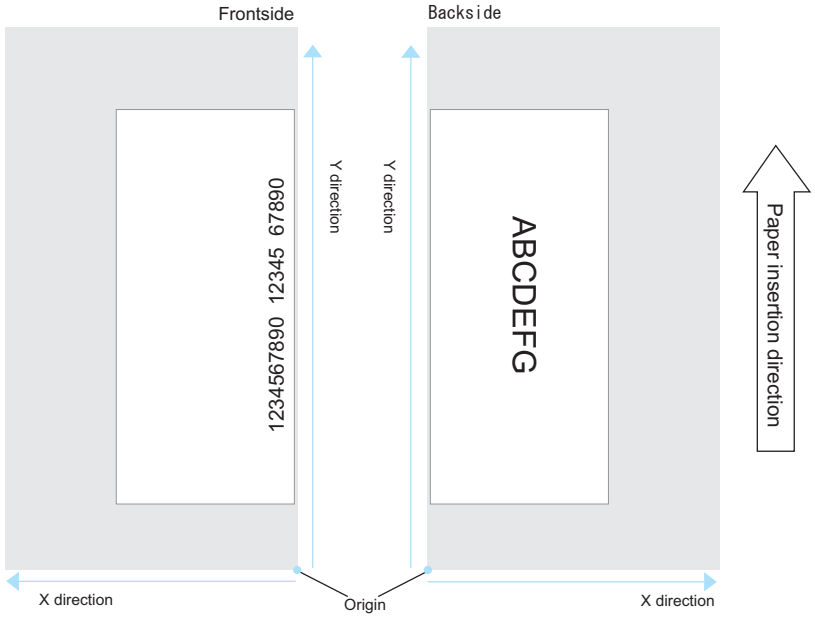
bEndY: Specifies the end Y coordinate (0 to 230) of the read area in units of mm.
This is a BYTE type. The initial value is 0.
When 0 is specified, the internal measured value of the device is specified.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The origin of the front face coordinates (0,0) corresponds the bottom-right corner of the check sheet, when oriented as if to be inserted. The origin of the rear face coordinates (0,0) corresponds the bottom-left corner of the check sheet, when oriented as if to be inserted. The set value remains valid until the device is closed.



Note

- ❏ When the start point is beyond the end point (Start >/= End), ERR_PARAM is returned to the return value.
- ❏ When a value exceeding the area that the device can read is specified, the maximum value within the area that the device can read is set.
- ❏ The read area set by this API is applied from the next read processing.
- ❏ If an odd number is specified, it is rounded to the nearest even number. The start point (*bStartX*,*bStartY*) is rounded down, and the end point (*bEndX*,*bEndY*) is rounded up to the nearest even number.

BiSCNGetImageQuality

Acquires the scanning quality of the image.

Syntax

int **BiSCNGetImageQuality**(*int nHandle, LPBYTE pColorDepth, char *pThreshold, LPBYTE pColor,LPBYTE pExOption)*

Argument

nHandle: Specifies the handle. This is an INT type.

pColorDepth: Specifies the memory address where the tonal gradation is set. This is an LPBYTE type.

Constant	Value	Description
EPS_BI_SCN_1BIT	1	1 bit
EPS_BI_SCN_8BIT	8	8 bit

pThreshold: Specifies the memory address where the density threshold value is set. This is a char type.

pColor: Specifies the memory address where color is set. The set value is as shown below. This is an LPBYTE type.

Constant	Value	Description
EPS_BI_SCN_MONOCHROME	48	Black and White
EPS_BI_SCN_COLOR	49	Color

pExOption: Specifies a memory address to which a density adjustment type is set. Set values are as below. This is an LPBYTE type.

Constant	Value	Description
EPS_BI_SCN_MANUAL	49	Density is adjusted manually. (Density adjustment is not executed by the driver.)
EPS_BI_SCN_SHARP	50	Sharpening
EPS_BI_SCN_SHARP_CUSTOM	51	Sharpening (for compatibility with TM-J9000 API. The setting is the same as EPS_BI_SCN_SHARP.)
EPS_BI_SCN_SHARP_CUSTOM2	52	Sharpening (for compatibility with TM-J9000 API. The setting is the same as EPS_BI_SCN_SHARP.)
EPS_BI_SCN_SHARP_CUSTOM3	53	Edge-preserving smoothing

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Check the acquired image scan quality referring to the table below.

bColorDepth	bExOption	Scan result
EPS_BI_SCN_1BIT	EPS_BI_SCN_MANUAL	A value set to bThreshold is applied
	EPS_BI_SCN_SHARP	bThreshold is automatically set, and the set value is applied
EPS_BI_SCN_8BIT	EPS_BI_SCN_MANUAL	Settings of only bColorDepth and bColor are applied
	EPS_BI_SCN_SHARP	Sharpening

BiSCNGetImageFormat

Acquires the format of the image.

Syntax

int **BiSCNGetImageFormat**(*int nHandle, LPBYTE pFormat)*

Argument

nHandle: Specifies the handle. This is an INT type.

pFormat: Specifies the memory address where the format of the notified image data is set. The set value is as shown below. This is an LPBYTE type.

Constant	Value	Description
EPS_BI_SCN_TIFF	1	TIFF format CCITT (Group 4) compressed data
EPS_BI_SCN_RASTER	2	Raster format uncompressed data
EPS_BI_SCN_BITMAP	3	Bitmap format uncompressed data
EPS_BI_SCN_TIFF256	4	TIFF format uncompressed data
EPS_BI_SCN_JPEGHIGH	5	JPEG format high compression (size priority) data
EPS_BI_SCN_JPEGNORMAL	6	JPEG format normal compression data
EPS_BI_SCN_JPEGLOW	7	JPEG format low compression (quality priority) data
EPS_BI_SCN_JTIFF	8	TIFF format JPEG compressed data

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

BiSCNGetScanArea

The read area of the device scanner image is acquired. This API is compatible with the TM-J9000.

Syntax

int **BiSCNGetScanArea** (*int nHandle, LPBYTE pStartX, LPBYTE pStartY, LPBYTE pEndX, LPBYTE pEndY*)

Argument

nHandle: Specifies the handle. This is an INT type.

bStartX: Returns the start X coordinate of the read area. This is a BYTE type.

bStartY: Returns the start Y coordinate of the read area. This is a BYTE type.

bEndX: Returns the end X coordinate of the read area. This is a BYTE type.

bEndY: Returns the end X coordinate of the read area. This is a BYTE type.

Return Value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error

Explanation

The origin of the surface coordinates (0,0) corresponds the bottom-right corner of the check sheet, when oriented as if to be inserted. The origin of the rear face coordinates (0,0) corresponds the bottom-left corner of the check sheet, when oriented as if to be inserted.

BiSCNSetCroppingArea

Sets the cropping area. This API is compatible with the TM-J9000.

Syntax

int **BiSCNSetCroppingArea** (*int nHandle, BYTE bAreaNo, BYTE bStartX, BYTE bStartY,BYTE bEndX, BYTE bEndY*)

Argument

nHandle: Specifies the handle. This is an INT type.

bAreaNo: Specifies the cropping area number (1 to 255). This is a BYTE type.

bStartX: Specifies the start X coordinate (0 to 98) of the cropping area in units of mm. This is a BYTE type.

bStartY: Specifies the start Y coordinate (0 to 228) of the cropping area in units of mm. This is a BYTE type.

bEndX: Specifies the end X coordinate (2 to 100) of the cropping area in units of mm. This is a BYTE type. When a value that exceeds the area that can be cropped, the maximum value of the croppable area is set.

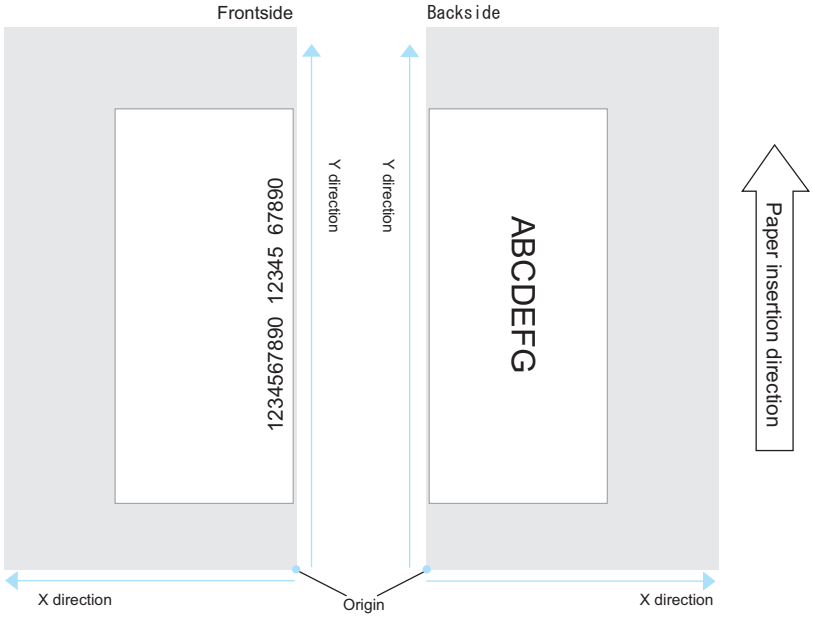
bEndY: Specifies the end Y coordinate (2 to 230) of the read area in units of mm. This is a BYTE type. When a value that exceeds the area that can be cropped, the maximum value of the croppable area is set.

Return Value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The origin of the surface coordinates (0,0) corresponds the bottom-right corner of the check sheet, when oriented as if to be inserted. The origin of the rear face coordinates (0,0) corresponds the bottom-left corner of the check sheet, when oriented as if to be inserted. The set value remains valid until the device is closed.



Note

- ❏ When the start point is beyond the end point (Start >/= End), ERR_PARAM is returned to the return value.
- ❏ When the specified Cropping area number already exists, set a new Cropping area.
- ❏ When a value that exceeds the area that can be cropped for the device, the maximum value of the area that the device can crop is set.
- ❏ The read area set by this API is applied from the next read processing.
- ❏ If an odd number is specified, it is rounded to the nearest even number. The start point (bStartX, bStartY) is rounded down, and the end point (bEndX, bEndY) is rounded up to the nearest even number.

BiSCNGetCroppingArea

Acquires cropping area information from the device. This API is compatible with the TM-J9000.

Syntax

int **BiSCNGetCroppingArea** (*int nHandle, LPWORD pBuffSize, LPBYTE pBuff*)

Argument

nHandle: Specifies the handle. This is an INT type.

pBuffSize: Specifies the size of the memory in which the cropping area information is set. After the BiSCNGetCroppingArea designation has been issued, the size of the actually read data is returned. When the memory size specified in pBuffSize is insufficient, the required memory size is returned. Nothing is returned in the event of any other error. This is an LPWORD type.

pBuff: Specifies the memory address that acquires the cropping area information. This is an LPBYTE type.

Return Value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-140	ERR_BUFFER_OVER_FLOW	Buffer overflow error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

pBuff acquires the cropping area information in the format as shown below.

Cropping area number

Start X coordinate of the cropping area

Start Y coordinate of the cropping area

End X coordinate of the cropping area

End Y coordinate of the cropping area

BiSCNDeleteCroppingArea

Deletes a registered cropping area. This API is compatible with the TM-J9000.

Syntax

int **BiSCNDeleteCroppingArea** (*int nHandle, BYTE AreaNo*)

Argument

nHandle: Specifies the handle. This is an INT type.

bAreaNo: Specifies the number (0 to 255) of the cropping area to be deleted. This is a BYTE type.

Return Value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

When 0 is specified for bAreaNo, the entire cropping area is deleted.

BiSCNSelectScanUnit

Sets the unit that operates image scanning. This API is compatible with the TM-J9000.

Syntax

int **BiSCNSelectScanUnit***(int nHandle, BYTE bSelectUnit)*

Argument

nHandle: Specifies the handle. This is an INT type.

bSelectUnit: Specifies the unit to scan. The selectable value is as follows

Constant	Value	Description
EPS_BI_SCN_UNIT_CHECKPAPER	48	Check reading unit

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Sets EPS_BI_SCN_UNIT_CHECKPAPER for the initial value. The functions below are effective to the unit currently selected.

- BiSCNSetImageQuality
- BiSCNSetScanArea
- BiSCNSetImageFormat
- BiSCNSetCroppingArea
- BiSCNGetImageQuality
- BiSCNGetScanArea
- BiSCNGetCroppingArea

BiSCNMICRFunction

TM-J9000-compatible API that performs reading with the scanner. Use of BiSCNMICRFunctionContinuously or BiSCNMICRFunctionPostPrint is recommended.

Syntax

int **BiSCNMICRFunction***(int nHandle, LPVOID lpvStruct, WORD wFunction)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpvStruct: The address of the parameter structure specified for each unit. This is an LPVOID type.

wFunction: Specifies the functions for BiSCNMICRFunction to execute. Reading-related settings are notified to StatusAPI by specifying MF_SET_MICR_PARAM, etc. and executing wFunction. If the members of the structure are changed after the setting notification, another notification with by using wFunction is required. Include the header file submitted in the definition name used for BiSCNMICRFunction. This is a WORD type.

lpvStruct supplementary explanation
Refer to "BiSCNMICRFunctionContinuously" on page 4-58.

Definition of function settings
Refer to "BiSCNMICRFunctionContinuously" on page 4-58.

Initial values set to each of the structures
Refer to "BiSCNMICRFunctionContinuously" on page 4-58.

Return value

BiSCNMICRFunction

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	Cannot be used due to waiting for online reset
-300	ERR_PAPERINSERT_TIMEOUT	Paper insertion time exceeded
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected

BiSCNMICRFunction

Value	Constant	Description
-440	ERR_MICR	Printer failed in MICR reading
-450	ERR_SCAN	Printer failed in image scanning
-1020	ERR_PAPER_JAM	Paper jam has occurred
-1100	ERR_PAPER_INSERT	Paper insertion error

MF_BASE.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-300	ERR_PAPERINSERT_TIME OUT	Paper insertion time exceeded
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-440	ERR_MICR	Printer failed in MICR reading
-450	ERR_SCAN	Printer failed in image scanning
-470	ERR_NOT_EXEC	Process not being executed
-1020	ERR_PAPER_JAM	Paper jam has occurred
-1100	ERR_PAPER_INSERT	Paper insertion error

MF_SCAN.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-450	ERR_SCAN	Printer failed in image scanning
-470	ERR_NOT_EXEC	Process not being executed

MF_MICR.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-140	ERR_BUFFER_OVER_FLOW	Buffer overflow error
-220	ERR_NOT_FOUND	No data error
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-440	ERR_MICR	Printer failed in MICR reading
-470	ERR_NOT_EXEC	Process not being executed

MF_BARCODE.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-90	ERR_PARAM	Parameter error
-220	ERR_NOT_FOUND	No data error
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-470	ERR_NOT_EXEC	Process not being executed
-1130	ERR_BARCODE_NODATA	Barcode cannot be detected.

MF_PRINT.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-470	ERR_NOT_EXEC	Process not being executed

Explanation

Specify the structure for the parameters of the function you wish to use among SCAN, MICR, and Transaction printing to call BiSCNMICRFunction. Be sure to set MF_SET_BASE_PARAM. To set the parameter structure again, change the members of the parameter structure and specify MF_SET_ xxxx_PARAM again to call BiSCNMICRFunction. The entire information on the structure is stored with StatusAPI until BiCloseMonPrinter is executed.

Specifying MF_EXEC for the third parameter executes the specified function. BiSCNMICRFunction performs in order of SCAN, MICR, and Transaction. As you are to set the return value to the parameter structure, do not scrap the structure. If you still scrap the structure, do so after calling MF_CLEAR_xxxx_PARAM.

Returns ERR_PARAM (Not ERR_NOT_SUPPORT) if any unsupported functions are specified.

Progress Status Messages

Message	wParam	lParam	Description
WM_MF_DONE	-	Return value	Ends BiSCNMICRFunction.
WM_MF_PROGRESS	MF_PHASE_INIT	MF_PROGRESS_START	Starts initialization.
		MF_PROGRESS_WAIT_PAPER	Waiting for the paper to be inserted. This message is not sent when the paper is already set.
		MF_PROGRESS_CLUMP_PAPER	The paper is clamped and is being fed.
		MF_PROGRESS_PAPER_PILED	Detected double feeding.
		MF_PROGRESS_DONE	The device has finished scanning /MICR reading.
	MF_PHASE_SCAN	Progress status of reading the table (%)	MF_SCAN_FACE_FRONT is set to the lower byte of the upper WORD, and the percentage is set to the lower byte of the lower WORD. 0 % is notified when reading has started, and 100 % is notified when reading has finished.
		Progress status of reading the rear face (%)	MF_SCAN_FACE_BACK is set to the lower byte of the upper WORD, and the percentage is set to the lower byte of the lower WORD. 0 % is notified when reading has started, and 100 % is notified when reading has finished.
	MF_PHASE_MICR	MF_PROGRESS_START	MIC_OCR reading has started.
		MF_PROGRESS_DONE	MIC_OCR reading has finished.
	MF_PHASE_BARCODE	Progress status of reading the table (%)	MF_SCAN_FACE_FRONT is set to the lower byte of the upper WORD, and the percentage is set to the lower byte of the lower WORD. 0 % is notified when reading has started, and 100 % is notified when reading has finished.
		Progress status of reading the rear face (%)	MF_SCAN_FACE_BACK is set to the lower byte of the upper WORD, and the percentage is set to the lower byte of the lower WORD. 0 % is notified when reading has started, and 100 % is notified when reading has finished.
	MF_PHASE_PRINT	MF_PROGRESS_START	Transaction printing has started.
		MF_PROGRESS_DONE	Transaction printing has finished.
	MF_PHASE_EXIT	MF_PROGRESS_START	Post-processing of paper edjection, etc. has started.
		MF_PROGRESS_DONE	Post-processing of paper edjection, etc. has finished.

* With WM_MF_PROGRESS, reading unit numbers are set to LOBYTE of HIWORD in wParam. MF_PHASE_INIT and other phase numbers are set in LOBYTE of LOWORD. For example, the reading unit number for check paper is EPS_BI_SCN_UNIT_CHECKPAPER.

* The contents of wParam and lParam on reception of WM_MF_PROGRESS can be obtained with the macros provided below.

Macro	wParam / lParam	Description
MF_MACRO_GETUNITID	wParam	For reading the unit number
MF_MACRO_GETPHASE	wParam	For obtaining the phase number
MF_MACRO_GETFACE	lParam	For obtaining the scanned image face
MF_MACRO_PERCENT	lParam	For obtaining the image scanning percentage

Note

1. When use of OCR is specified (with MF_MICR_USE_OCR bit of bMicOcrSelect of MF_MICR structure is set to ON) in the MICR setting, returns ERR_PARAM on execution of MF_EXEC if the scanning parameter for surface scanning is not set. This is because ORC processing requires the scan image of the front face.

2. When ElectricEndorsement processing is enabled (with blElectricEndorse of MF_PRINT structure is set to TRUE) in the transaction printing setting, returns ERR_PARAM on execution of MF_EXEC if the paramaters for endosement scanning is not set. This is because ElectricEndorsement processing requires the scan image of the rear face. When the ElectricEndorsement processing is enabled, no information is set to bStatus, bDetail, dwXSize, dwYSize, dwScanSize, or lpbScanData of the rear face scanning parameters until the transaction printing has finished even if scanning the rear face has succeeded.

3. Among the member variables for the MF_xxx structure, give special attention to the ones that stores address values. If an invalid address value is specified and executed this function, an application error may result. Therefore, be sure to specify NULL if specifying an address is unnecessary.

4. Among the member variables for the MF_xxx structure, give special attention to the ones that stores character strings. If this function is specified without including '\0' that indicates the final character string in the variables, an application error may result. Therefore, be sure to include '\0' in the variables.

5. If MF_BASE_MESSAGE_NO_MESSAGE is set for the dwNotifyType member variables of the MF_BASE01 structure and MF_EXEC of this function is executed (MF_EXEC of this function is executed synchronously), double feeding detection does not function.

BiSCNMICRCancelFunction

BiSCNMICRFunctionContinuously is interrupted.

Syntax

int **BiSCNMICRCancelFunction**(*int nHandle, WORD wEjectType)*

Argument

nHandle: Specifies the handle. This is an INT type.

wEjectType: This specifies the method for discharging paper. This is an LPDWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	Cannot be used because of off-line return wait
-130	ERR_WITHOUT_CB	BiSCNMICRFunctionPostPrint/ BiSCNMICRFunctionContinuously has not been run
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed

Explanation

BiSCNMICRFunctionContinuously is interrupted. Alternatively, paper-eject is executed after the ERR_LINE_OVERFLOW error occurs with the BiSCNMICRFunctionContinuously.
When interrupting the BiSCNMICRFunctionContinuously, if the interruption is not complete after a maximum of 20 seconds has elapsed, ERR_TIMEOUT is returned Once interrupted, the results are set in the storage area for the structure specified by BiSCNMICRFunctionContinuously.
The only times that ERR_EXEC_FUNCTION is returned by this API is when there are overlapping calls of this API with multiple threads and when the device is initialized after print setup and turning the device power off and on.

Note

Even if MF_EJECT_RELEASE is specified as the parameter of this API, paper is not ejected to the sub pocket. It is ejected to the main pocket.

BiSCNSelectScanFace

Specifies the back or front for an image that is being read.

Syntax

int **BiSCNSelectScanFace***(int nHandle, BYTE bFace)*

Argument

nHandle: Specifies the handle. This is an INT type.

bFace: This specifies the back or front for an image that is being read. This is a BYTE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

MF_SCAN_FACE_FRONT is specified for the front and MF_SCAN_FACE_BACK for the back. If values other than these are set, ERR_PARAM is returned.

BiGetPrnCapability

Obtains the device information specified by the device ID.

Syntax

int **BiGetPrnCapability** *(int nHandle, BYTE prnID, LPBYTE pBuffSize, LPBYTE pBuff)*

Argument

nHandle: Specifies the handle. This is an INT type.

prnID: Specifies the device ID from which obtain information. See "Device information" on page 4-1 for details on Device ID.

pBuffSize: Specifies the size of the memory to which the device information is stored. Values of 1 - 80 can be specified. After pBuffSize is run, sets the actual size of the read data. If the actual data size exceeds the specified data size, notifies the operator of the actual data size to set again. If any other errors occur, sets no value.

pBuff: Specifies the address of the memory to which the device information is stored.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	Cannot be used because the device is waiting for online reset.
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Specifying an unsupported device ID and issuing this command results in either of the following. The result depends on the product and firmware.

- Time error occurs (Return value: ERR_TIMEOUT).
- Returns "Success" (Return value: SUCCESS), and sets a value to the parameter pBuff. However, Epson does not guarantee the obtained pBuff parameter value.

See Device information for details on Device ID.

BiCloseMonPrinter

This closes the device that is undergoing status monitoring.

Syntax

int **BiCloseMonPrinter** *(int nHandle)*

Argument

nHandle: Specifies the handle. This is an INT type.

Return value

If closing of the device was successful, a 0 is returned. If there is an error, the following error code (less than zero) is returned.

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect

BiGetRealStatus

Acquires the current device status. The device status is set to the lpStatus. This API is compatible with the TM-J9000.

Syntax

int **BiGetRealStatus** (*int nHandle, LPDWORD lpStatus*)

Argument

nHandle: Specifies the handle. This is an INT type.

lpStatus: The current status of the device is set. This is an LPDWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Refer to "Device Status" on page 4-1, for the device statuses that you can acquire.

BiSCNMICRFunctionContinuously

Scans images successively, and reads MICR characters.

Syntax

int **BiSCNMICRFunctionContinuously***(int nHandle, LPVOID lpvStruct, WORD wFunction)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpvStruct: The address of the parameter structure specified for each unit. This is an LPVOID type.

wFunction: Specifies the functions for the API to execute. For settings regarding reading, by specifying MF_SET_MICR_PARAM and so on and executing this function, the API is notified. If the members of the structure are changed after notification, it is necessary to perform notification again with this function. Include the header file submitted for the definition name used for this API. This is a WORD type.

lpvStruct supplementary explanation
For an explanation of the lpvStruct structure, refer to "Structures" on page 4-138 However, the following values are not used with this API, or the values are not set.

<MF_BASE structure>

- dwNotifyType and uNotifyHandle are not used
  Notification of the reading status of this API is performed by the handler registered with BiSCNMICRSetStatusBackFunction.

- hProgressWnd is not used
  This API does not provide notification of the progress status.

<MF_SCAN structure>

- wImageID is not used
  Set the transaction number (ID) using BiSetTransactionNumber

- Do not set values in bStatus, bDetail, dwXSize, dwYSize, dwScanSize, and lpbScanData
  Set values when getting the scan image with BiGetScanImage.

<MF_MICR structure>

- bMicOcrSelect and blParsing are not used
  Use them when executing BiGetMICRText.

- Do not set values in bStatus, bDetail, szMicrStr, stOcrReliableInfo, szAccountNumber, szAmount, szBankNumber, szSerialNumber, szEPC, szTransitNumber, lCheckType, and lCountryCode
  Set values when getting the MICR (OCR) text with BiGetMICRText.

Supplemental Description for wFunction

Constant	Description
MF_EXEC	SCAN / MICR / Transaction printing is carried out according to the specified parameters. The second parameter is ignored.
MF_CONTINUE	ERR_NOT_SUPPORT is returned.
MF_MICR_RETRANS	ERR_NOT_SUPPORT is returned.
MF_SCAN_FRONT_RETRANS	ERR_NOT_SUPPORT is returned.
MF_SCAN_BACK_RETRANS	ERR_NOT_SUPPORT is returned.

Definitions for Function Settings

Constant	Description
MF_SET_BASE_PARAM	This sets base parameters. The MF_BASE structure address is specified in lpvStruct.
MF_SET_MICR_PARAM	This sets MICR parameters. The MF_MICR structure address is specified in lpvStruct.
MF_SET_SCAN_FRONT_PARAM	This sets front side read scanning parameters. The MF_SCAN structure address is specified in lpvStruct. The MF_SCAN structure has a different address from the structure for reading the back side. When the same address is specified, ERR_PARAM is returned. The operation is identical when MF_SET_SCAN_PARAM is specified.
MF_SET_SCAN_BACK_PARAM	This sets the back side read scanning parameters. The MF_SCAN structure address is specified in lpvStruct. The MF_SCAN structure has a different address from the structure for reading the front side. When the same address is specified, ERR_PARAM is returned.
MF_SET_PRINT_PARAM	This sets transaction printing parameters. The MF_PRINT01 structure address is specified in lpvStruct.
MF_SET_PROCESS_PARAM	This sets process parameters. The MF_PROCESS structure address is specified in lpvStruct.
MF_SET_IQA_PARAM	This sets IQA parameters. The MF_IQA structure address is specified in lpvStruct.
MF_SET_BARCODE_FRONT_PARAM	This sets the front side barcode decode parameters. The MF_BARCODE structure address is specified in lpvStruct.
MF_SET_BARCODE_BACK_PARAM	This sets the back side barcode decode parameters. The MF_BARCODE structure address is specified in lpvStruct.
MF_CLEAR_BASE_PARAM	This clears all the specifications for BASE / MICR / SCAN / PRINT/ PROCESS parameters. lpvStruct values are ignored.
MF_CLEAR_MICR_PARAM	This clears the MICR parameter specifications. lpvStruct values are ignored.
MF_CLEAR_SCAN_FRONT_PARAM	This clears the scan parameter specifications. lpvStruct values are ignored. The operation is identical when MF_CLEAR_SCAN_PARAM is specified.
MF_CLEAR_SCAN_BACK_PARAM	This clears the scan parameter specifications. lpvStruct values are ignored.
MF_CLEAR_PRINT_PARAM	This clears the transaction printing parameter specifications. lpvStruct values are ignored.

Definitions for Function Settings

Constant	Description
MF_CLEAR_PROCESS_PARAM	This clears the process parameter specifications. lpvStruct values are ignored.
MF_CLEAR_IQA_PARAM	This clears the IQA parameter specifications. lpvStruct values are ignored.
MF_CLEAR_BARCODE_FRONT_PARAM	This clears the front side barcode decode parameters specifications. lpvStruct values are ignored.
MF_CLEAR_BARCODE_BACK_PARAM	This clears the back side barcode decode parameters specifications. lpvStruct values are ignored.
MF_GET_BASE_DEFAULT	This obtains the initial values for the device base structure.
MF_GET_MICR_DEFAULT	This obtains the initial values for the device MICR structure.
MF_GET_SCAN_DEFAULT	This obtains the initial values for the device SCAN structure.
MF_GET_SCAN_FRONT_DEFAULT	This obtains the initial values for the device SCAN (front side) structure.
MF_GET_SCAN_BACK_DEFAULT	This obtains the initial values for the device SCAN (back side) structure.
MF_GET_PRINT_DEFAULT	This obtains the initial values for the transaction printing structure. * With API, the initial values of iSize and iVersion are not returned. The application must specify these two values. Also, the initial value for the structure must be obtained after zero clearing all member variables except iSize and iVersion.
MF_GET_PROCESS_DEFAULT	This obtains the initial values for the process structure.
MF_GET_IQA_DEFAULT	This obtains the initial values for the IQA structure.
MF_GET_BARCODE_FRONT_DEFAULT	This obtains the initial values for the device Barcode (front side) structure.
MF_GET_BARCODE_BACK_DEFAULT	This obtains the initial values for the device Barcode (back side) structure.

Return value

BiSCNMICRFunction

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-110	ERR_OFFLINE	Cannot be used due to waiting for online reset
-130	ERR_WITHOUT_CB	Cannot be executed as neither of BiSCNMICRSetStatusBackFunction is invoked
-300	ERR_PAPERINSERT_TIMEOUT	Paper insertion time exceeded
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-440	ERR_MICR	Printer failed in MICR reading
-450	ERR_SCAN	Printer failed in image scanning
-460	ERR_LINE_OVERFLOW	Line overflow occurred during transaction printing
-1010	ERR_PAPER_PILED	Paper pilling error
-1020	ERR_PAPER_JAM	Paper jam has occurred
-1030	ERR_COVER_OPEN	Cover open error
-1040	ERR_MICR_NODATA	MICR data is not existing
-1050	ERR_MICR_BADDATA	MICR data is not able to recognize
-1070	ERR_MICR_NOISE	Noise error has occurred during MICR reading
-1080	ERR_SCN_COMPRESS	Scan image data compressing error
-1090	ERR_PAPER_EXIST	API can not be execute because there is a paper on the path
-1100	ERR_PAPER_INSERT	Paper insertion error

MF_BASE.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-300	ERR_PAPERINSERT_TIMEOUT	Paper insertion time exceeded
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-440	ERR_MICR	Printer failed in MICR reading
-450	ERR_SCAN	Printer failed in image scanning
-460	ERR_LINE_OVERFLOW	Line overflow occurred during transaction printing
-470	ERR_NOT_EXEC	Process not being executed
-1010	ERR_PAPER_PILED	Paper pilling error
-1020	ERR_PAPER_JAM	Paper jam has occurred
-1030	ERR_COVER_OPEN	Cover open error
-1040	ERR_MICR_NODATA	MICR data is not existing
-1050	ERR_MICR_BADDATA	MICR data is not able to recognize
-1070	ERR_MICR_NOISE	Noise error has occurred during MICR reading
-1080	ERR_SCN_COMPRESS	Scan image data compressing error
-1090	ERR_PAPER_EXIST	API can not be execute because there is a paper on the path
-1100	ERR_PAPER_INSERT	Paper insertion error

MF_SCAN.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-450	ERR_SCAN	Printer failed in image scanning
-470	ERR_NOT_EXEC	Process not being executed
-1080	ERR_SCN_COMPRESS	Scan image data compressing error
-1090	ERR_PAPER_EXIST	API can not be execute because there is a paper on the path
-1100	ERR_PAPER_INSERT	Paper insertion error

MF_MICR.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-140	ERR_BUFFER_OVER_FLOW	Buffer overflow error
-220	ERR_NOT_FOUND	No data error
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-440	ERR_MICR	Printer failed in MICR reading
-470	ERR_NOT_EXEC	Process not being executed
-1040	ERR_MICR_NODATA	MICR data is not existing
-1050	ERR_MICR_BADDATA	MICR data is not able to recognize
-1070	ERR_MICR_NOISE	Noise error has occurred during MICR reading

MF_BARCODE.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-90	ERR_PARAM	Parameter error
-220	ERR_NOT_FOUND	No data error
-470	ERR_NOT_EXEC	Process not being executed
-1130	ERR_BARCODE_NODATA	Barcode cannot be detected.

MF_PRINT.iRet

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible
-430	ERR_ABORT	Canceled by BiSCNMICRCancelFunction or near full is detected
-460	ERR_LINE_OVERFLOW	Line overflow occurred during transaction printing
-470	ERR_NOT_EXEC	Process not being executed

MF_IQA_RESULT.iRet

Value	Constant	Description
0	SUCCESS	Success
-450	ERR_SCAN	Printer failed in image scanning
-1120	ERR_SCN_IQA	Error is detected by the IQA validation.

Explanation

From the SCAN, MICR, or Ttransaction printing, specifies the parameters of the functions wanted to be used by the structure and invokes this API. It is required that MF_SET_BASE_PARAM is set. When resetting the structure, change the members of the structure and invoke this API again specifying MF_SET_xxxx_PARAM. The contents of all the structures are stored by TM-S1000 API until BiCloseMonPrinter is executed.

By specifying MF_EXEC as the 3rd parameter, the specified functions are executed for all check paper inserted in the feeder. When the feeder is empty, it stops automatically. Be sure not to discard the structure in order to set the return value to the structure. Be sure to invoke MF_CLEAR_xxxx_PARAM before discarding the structure.

However, scan image data and MICR text is not set for dwXSize, dwYSize, dwScanSize, and lpbScanData of the MF_SCAN structure, and szMicrStr, stOcrReliableInfo, szAccountNumber, szAmount, szBankNumber, szSerialNumber, szEPC, szTransitNumber, lCheckType, and lCountryCode of the MF_MICR structure. After MF_DATARECEIVE_DONE notification, by specifying and invoking the TransactionNumber that corresponds to BiGetScanImage and BiGetMicrText, the scan image data and MICR text can be acquired.

The TM-S1000 stops scanning operation when 10 scanned-in images are stored in the driver. The TM-S1000 resumes the scanning operation when the number of stored images become two or less.

Processing status list

Main status	Sub status	Outline
MF_FUNCTION_START	-	Started check paper scanning
MF_CHECKPAPER_PROCESS_START	-	Inserted check paper and started scanning
MF_DATARECEIVE_START	-	Started receiving data
MF_DATARECEIVE_DONE	-	Completed receiving data
MF_CHECKPAPER_PROCESS_DONE	-	Ejected check paper and completed the process
MF_FUNCTION_DONE	iRet of the MF_BASE structure	Finished check paper scanning
MF_ERROR_OCCURED	ERR_PAPER_PILED	Detected double feed (Reading result: 47H)
	ERR_FORM_LENGTH	A paper jam error occurred (Reading result details: 44H, 45H)
	ERR_PAPER_JAM	A paper jam error occurred (Reading result details: 46H)
	ERR_MECHANICAL	A mechanical error has occurred in the middle of scanning. (Reading result details: 47H)
	ERR_COVER_OPEN	Processing has stopped because of the cover open. (Reading result details: 48H)
	ERR_MICR_NODATA	Magnetic waveform detection error (MICR details: 45H)
	ERR_MICR_BADDATA	Characters unable to be analyized detection error (MICR details: 46H)
	ERR_MICR_NOISE	Noise error(MICR details: 47H)
	ERR_SCN_COMPRESS	Data compression error (SCN details: 47H)
	ERR_SCN_IQA	NOT_PASS is detected at the IQA validation.
	ERR_BARCODE_NODATA	Barcode cannot be detected.

Default Settings

MF_GET_BASE_DEFAULT	MF_BASE01.dwNotifyType = MF_BASE_MESSAGE_HWND
	MF_BASE01.dwTimeout = MF_BASE_TIMEOUT_DEFAULT
	MF_BASE01.uNotifyHandle.hNotifyWnd = 0
	MF_BASE01.hProgressWnd = 0
	MF_BASE01.wErrorEject = MF_EXIT_ERROR_RELEASE
	MF_BASE01.bBuzzerHz(0) = MF_BUZZER_HZ_4000
	MF_BASE01.bBuzzerHz(1) = MF_BUZZER_HZ_4000
	MF_BASE01.bBuzzerHz(2) = MF_BUZZER_HZ_4000
	MF_BASE01.bBuzzerCount(0) = MF_BUZZER_DISABLE
	MF_BASE01.bBuzzerCount(1) = MF_BUZZER_DISABLE
	MF_BASE01.bBuzzerCount(2) = MF_BUZZER_DISABLE
	MF_BASE01.bUseNVMemory = MF_BASE_NVMEMORY_NOT_USE
	MF_BASE01.wSuccessEject = MF_EXIT_SUCCESS_DISCHARGE
MF_GET_MICR_DEFAULT	MF_MICR.bFont = MF_MICR_FONT_E13B
	MF_MICR.bMicOcrSelect = MF_MICR_USE_MICR
	MF_MICR.blParsing = FALSE
	MF_MICR.bStatus = 0
	MF_MICR.bDetail = 0
	MF_MICR.szMicrStr : zero clear
	MF_MICR.stOcrReliableInfo : zero clearzero clear
	MF_MICR.szAccountNumber : zero clear
	MF_MICR.szAmount : zero clear
	MF_MICR.szBankNumber : zero clear
	MF_MICR.szSerialNumber : zero clear
	MF_MICR.szEPC : zero clear
	MF_MICR.szTransitNumber : zero clear
	MF_MICR.lCheckType = 0
	MF_MICR.lCountryCode = 0
MF_GET_SCAN_DEFAULT MF_GET_SCAN_FRONT_DEFAULT MF_GET_SCAN_BACK_DEFAULT	MF_SCAN.wImageID = 1
	MF_SCAN.sResolution = MF_SCAN_DPI_DEFAULT
	MF_SCAN.bAddInfoDataSize = 0
	MF_SCAN.pAddInfoData = NULL
	MF_SCAN.bStatus = 0
	MF_SCAN.bDetail = 0

Default Settings

MF_GET_SCAN_DEFAULT MF_GET_SCAN_FRONT_DEFAULT MF_GET_SCAN_BACK_DEFAULT	MF_SCAN.dwXSize = 0
	MF_SCAN.dwYSize = 0
	MF_SCAN.dwScanSize = 0
	MF_SCAN.lpbScanData = NULL
MF_GET_PROCESS_DEFAULT	MF_PROCESS.bActivationMode = MF_ACTIVATE_MODE_HIGH_SPEED
	MF_PROCESS.bPaperType = MF_PAPER_TYPE_CHECK
	MF_PROCESS.wSendPaperASF = 0
	MF_PROCESS.dwStartWaitTime = 1000
	MF_PROCESS.bSuccessStamp = MF_STAMP_DISABLE
	MF_PROCESS.bPaperMisInsertionErrorSelect= MF_ERROR_SELECT_DETECT
	MF_PROCESS.bPaperMisInsertionErrorEject= MF_EJECT_MAIN_POCKET
	MF_PROCESS.bPaperMisInsertionStamp = MF_STAMP_DISABLE
	MF_PROCESS.bPaperMisInsertionCancel = MF_CANCEL_DISABLE
	MF_PROCESS.bNoiseErrorSelect = MF_ERROR_SELECT_DETECT
	MF_PROCESS.bNoiseErrorEject = MF_EJECT_MAIN_POCKET
	MF_PROCESS.bNoiseStamp = MF_STAMP_DISABLE
	MF_PROCESS.bNoiseCancel = MF_CANCEL_ENABLE
	MF_PROCESS.bDoubleFeedErrorSelect = MF_ERROR_SELECT_DETECT
	MF_PROCESS.bDoubleFeedErrorEject = MF_EJECT_MAIN_POCKET
	MF_PROCESS.bDoubleFeedStamp = MF_STAMP_DISABLE
	MF_PROCESS.bDoubleFeedCancel = MF_CANCEL_DISABLE
	MF_PROCESS.bBaddataErrorSelect = MF_ERROR_SELECT_DETECT
	MF_PROCESS.bBaddataCount = 255
	MF_PROCESS.bBaddataErrorEject = MF_EJECT_MAIN_POCKET
	MF_PROCESS.bBaddataStamp = MF_STAMP_DISABLE
	MF_PROCESS.bBaddataCancel = MF_CANCEL_DISABLE
	MF_PROCESS.bNodataErrorSelect = MF_ERROR_SELECT_DETECT
	MF_PROCESS.bNodataErrorEject = MF_EJECT_MAIN_POCKET
	MF_PROCESS.bNodataStamp = MF_CANCEL_DISABLE
	MF_PROCESS.bNodataCancel = MF_CANCEL_DISABLE
	MF_PROCESS.bNearFullSelect = MF_NEARFULL_PERMIT
	MF_PROCESS.bResultPartialData = MF_RESULT_NONE
MF_GET_PRINT_DEFAULT	MF_PRINT01.blDummy = FALSE
	MF_PRINT01.lpString(0) = NULL
	MF_PRINT01.lpString(1) = NULL
	MF_PRINT01.lpString(2) = NULL

Default Settings

MF_GET_PRINT_DEFAULT	MF_PRINT01.dwAttribute(0) = MF_PRINT_NO_ATTRIBUTE
	MF_PRINT01.dwAttribute(1) = MF_PRINT_NO_ATTRIBUTE
	MF_PRINT01.dwAttribute(2) = MF_PRINT_NO_ATTRIBUTE
	MF_PRINT01.wFont(2) = MF_PRINT_FONT_A
	MF_PRINT01.wFont(1) = MF_PRINT_FONT_A
	MF_PRINT01.wFont(2) = MF_PRINT_FONT_A
	MF_PRINT01.wFontSize(0) = MF_PRINT_FONT_W1_H1
	MF_PRINT01.wFontSize(1) = MF_PRINT_FONT_W1_H1
	MF_PRINT01.wFontSize(2) = MF_PRINT_FONT_W1_H1
	MF_PRINT01.bSpeed = MF_PRINT_SPEED_HIGH
	MF_PRINT01.bDirection = MF_PRINT_DIRECTION_DOUBLE
	MF_PRINT01.dwEndorseType = MF_PRINT_TYPE_ELECTRIC_ENDORSE_EXTEND
MF_GET_IQA_DEFAULT	MF_IQA.bErrorSelect = MF_ERROR_SELECT_NODETECT
	MF_IQA.bErrorEject = MF_EJECT_MAIN_POCKET
	MF_IQA.bStamp = MF_STAMP_DISABLE
	MF_IQA.bCancel = MF_CANCEL_DISABLE
	MF_IQA.bImageFormat = EPS_BI_SCN_TIFF
	MF_IQA.bColorDepth = EPS_BI_SCN_1BIT
	MF_IQA.bThreshold = 0
	MF_IQA.bColor = EPS_BI_SCN_MONOCHROME
	MF_IQA.bExOption = EPS_BI_SCN_SHARP_CUSTOM2
	MF_IQA.sResolution = MF_SCAN_DPI_DEFAULT
	MF_IQA.bUndersize = MF_IQA_TEST_DISABLE
	MF_IQA.bOversize = MF_IQA_TEST_DISABLE
	MF_IQA.bMincompressed = MF_IQA_TEST_DISABLE
	MF_IQA.bMaxcompressed = MF_IQA_TEST_DISABLE
	MF_IQA.bFront_rear = MF_IQA_TEST_DISABLE
	MF_IQA.bToolight = MF_IQA_TEST_DISABLE
	MF_IQA.bToodark = MF_IQA_TEST_DISABLE
	MF_IQA.bStreaks = MF_IQA_TEST_DISABLE
	MF_IQA.bNoise = MF_IQA_TEST_DISABLE
	MF_IQA.bFocus = MF_IQA_TEST_DISABLE
	MF_IQA.bCorners = MF_IQA_TEST_DISABLE
	MF_IQA.bEdges = MF_IQA_TEST_DISABLE
	MF_IQA.bFraming = MF_IQA_TEST_DISABLE

Default Settings

MF_GET_IQA_DEFAULT	MF_IQA.bSkew = MF_IQA_TEST_DISABLE
	MF_IQA.bCarbon = MF_IQA_TEST_DISABLE
	MF_IQA.bPiggyback = MF_IQA_TEST_DISABLE
MF_GET_BARCODE_FRONT_DEFAULT MF_GET_BARCODE_BACK_DEFAULT	MF_BARCODE.bErrorSelect = MF_ERROR_SELECT_NODETECT
	MF_BARCODE.bErrorEject = MF_EJECT_MAIN_POCKET
	MF_BARCODE.bStamp = MF_STAMP_DISABLE
	MF_BARCODE.bCancel = MF_CANCEL_DISABLE
	MF_BARCODE.dwTargetColor= MF_BARCODE_TARGET_COLOR_GRAY
	MF_BARCODE.sResolution = MF_SCAN_DPI_DEFAULT
	MF_BARCODE.dwInfoMode= 0
	MF_BARCODE.stInfo(0).dwSymbolMask = 0
	MF_BARCODE.stInfo(0).bDirection = MF_BARCODE_DIRECTION_ALL
	MF_BARCODE.stInfo(0).bOrigin = MF_BARCODE_ORIGIN_TOP_LEFT
	MF_BARCODE.stInfo(0).wStartX = 0
	MF_BARCODE.stInfo(0).wStartY = 0
	MF_BARCODE.stInfo(0).wEndX = 0
	MF_BARCODE.stInfo(0).wEndY = 0
	MF_BARCODE.stInfo(1).dwSymbolMask = 0
	MF_BARCODE.stInfo(1).bDirection = MF_BARCODE_DIRECTION_ALL
	MF_BARCODE.stInfo(1).bOrigin = MF_BARCODE_ORIGIN_TOP_LEFT
	MF_BARCODE.stInfo(1).wStartX = 0
	MF_BARCODE.stInfo(1).wStartY = 0
	MF_BARCODE.stInfo(1).wEndX = 0
	MF_BARCODE.stInfo(1).wEndY = 0
	MF_BARCODE.stInfo(2).dwSymbolMask = 0
	MF_BARCODE.stInfo(2).bDirection = MF_BARCODE_DIRECTION_ALL
	MF_BARCODE.stInfo(2).bOrigin = MF_BARCODE_ORIGIN_TOP_LEFT
	MF_BARCODE.stInfo(2).wStartX = 0
	MF_BARCODE.stInfo(0).wStartY = 0
	MF_BARCODE.stInfo(2).wEndX = 0
	MF_BARCODE.stInfo(2).wEndY = 0

Default Settings

MF_GET_BARCODE_FRONT_DEFAULT MF_GET_BARCODE_BACK_DEFAULT	MF_BARCODE.stInfo(3).dwSymbolMask = 0
	MF_BARCODE.stInfo(3).bDirection = MF_BARCODE_DIRECTION_ALL
	MF_BARCODE.stInfo(3).bOrigin = MF_BARCODE_ORIGIN_TOP_LEFT
	MF_BARCODE.stInfo(3).wStartX = 0
	MF_BARCODE.stInfo(3).wStartY = 0
	MF_BARCODE.stInfo(3).wEndX = 0
	MF_BARCODE.stInfo(3).wEndY = 0
	MF_BARCODE.stInfo(4).dwSymbolMask = 0
	MF_BARCODE.stInfo(4).bDirection = MF_BARCODE_DIRECTION_ALL
	MF_BARCODE.stInfo(4).bOrigin = MF_BARCODE_ORIGIN_TOP_LEFT
	MF_BARCODE.stInfo(4).wStartX = 0
	MF_BARCODE.stInfo(4).wStartY = 0
	MF_BARCODE.stInfo(4).wEndX = 0
	MF_BARCODE.stInfo(4).wEndY = 0

Note

Before invoking this API, execute either of BiSCNMICRSetStatusBackFunction.

BiSCNMICRFunctionPostPrint

Scans images, and reads MICR characters.

Syntax

int **BiSCNMICRFunctionPostPrint** *(int nHandle, LPVOID lpvStruct, WORD wFunction)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpvStruct: Specifies the address of the parameter structure specified by each unit.

wFunction: Specifies the execution content for this API. For the read settings, specifying MF_SET_MICR_PARAM causes notification to be posted to the TM-S1000 API upon the execution of this function. If, after notification has been posted, there is a change to the members of the structure, notification is posted again using this function. To use the definition names for this API, include them in the provided vendor files.

lpvStruct Supplemental explanation

Refer to the following for an explanation of the lpvStruct structure.

Structure	Page
MF_BASE01	4-138
MF_MICR	4-142
MF_SCAN	4-146
MF_PRINT01	4-149
MF_PROCESS	4-152

The following values are not used by this API.

<MF_BASE01 Structure>

- dwNotifyType, uNotifyHandle
  Notification of the read status of this API is posted by the handler registered in BiSCNMICRSetStatusBackFunction/ BiSCNMICRSetStatusBackWnd.

- hProgressWnd
  This API does not post notification of progress status.

<MF_MICR Structure>

- bMicOcrSelect, blParsing
  Used when calling BiGetMicrText.

- No value is set in any of bStatus, bDetail, szMicrStr, stOcrReliableInfo, szAccountNumber, szAmount, szBankNumber, szSerialNumber, szEPC, szTransitNumber, lCheckType, or lCountryCode. A value is set when an MICR (OCR recognition) character string is acquired using BiGetMicrText.

<MF_SCAN Structure>

- wImageID
  Calls BiSetTransactionNumber and sets a transaction number (ID).

- No value is set in any of bStatus, bDetail, dwXSize, dwYSize, dwScanSize, or lpbScanData. A value is set when a scanned image is acquired using BiGetScanImage.

Explanation

For SCAN/MICR/transaction printing, use a structure to specify the parameters of the functions you wish to use and call them using this API. MF_SET_BASE_PARAM must always be set. When a structure is set again, change the members of the structure and then call this API again by specifying MF_SET_xxxx_PARAM. The structures contain all the results output by the TM-S1000 API, up until the execution of BiCloseMonPrinter.
By specifying MF_EXEC in the third parameter, the specified function is executed .
When MF_CONTINUE, MF_MICR_RETRANS, MF_SCAN_FRONT_RETRANS, and MF_SCAN_BACK_RETRANS are specified in the third parameter, ERR_NOT_SUPPORT is returned.
Because the return value is set in the structure, the structure must be deleted. In such a case, call MF_CLEAR_xxxx_PARAM.
After the processing has been returned from the handler that performs MF_DATARECEIVE_DONE notification processing to the TM-S1000 API, E-endorse is executed.

Processing status list

Refer to "BiSCNMICRFunctionContinuously" on page 4-58.

Note

Refer to "BiSCNMICRFunctionContinuously" on page 4-58.

Return value

Refer to "BiSCNMICRFunctionContinuously" on page 4-58.

BiSCNMICRSetStatusBackFunction

Registers callback functions for notification of the reading status.

Note

This is unavailable when the development environment is VB.

Syntax

int **BiSCNMICRSetStatusBackFunction**
(int nHandle, int (CALLBACK EXPORT pScnMicrCB)(DWORD dwTransactionNumber, WORD wMainStatus, WORD wSubStatus, LPSTR lpcPortName))*

Argument

nHandle: Specifies the handle. This is an INT type.

pScnMicrCB: Specifies the address of the callback functions for notification of the BiSCNMICRFunctionContinuously scanning processing status. This is a CALLBACK EXPORT.

Callback function parameters

dwTransactionNumber: The transaction number (ID) corresponding to the check paper of the processing status source. This is a DWORD type.

wMainStatus: The main status of the processing status. This is a WORD type.

wSubStatus: The sub status of the processing status. This is a WORD type.

lpcPortName: The memory address where the port name of the callback invoker is saved. This is an LPSTR type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Registers the address of the callback functions for notification of the BiSCNMICRFunctionContinuously / BiSCNMICRFunctionPostPrint scanning processing status. When the processing status of the check paper changes, the values are saved in dwTransactionNumber, wMainStatus, and wSubStatus, and the registered callback function is invoked. In this case, the port name is saved in lpcPortName in order to distinguish the callback invoker. For details of the processing status, refer to processing status in "BiSCNMICRFunctionContinuously" on page 4-58.

BiSCNMICRSetStatusBackWnd

Scans images, and reads MICR characters.

Syntax

int **BiSCNMICRSetStatusBackWnd** *(int nHandle, long hWnd,*
LPDWORD lpdwTransactionNumber,
LPWORD lpwMainStatus, LPWORD lpwSubStatus)

Argument

nHandle:	Specifies the handle. This is an INT type.
hWnd:	Specifies the window handle of a button that sends a click event to issue notification of the status of BiSCNMICRFunctionContinuously/ BiSCNMICRFunctionPostPrint processing.
lpdwTransactionNumber:	Specifies the memory address containing the transaction number (ID).
lpwMainStatus:	Specifies the memory address containing the main status for the processing.
lpwSubStatus:	Specifies the memory address containing the sub status for the processing.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Registers the memory address containing the value used to post notification of the handle of the button used to send a click event to indicate the processing status of BiSCNMICRFunctionContinuously/BiSCNMICRFunctionPostPrint. Do not delete the memory address registered with this API until the registration made with BiSCNMICRCancelStatusBack has been released. Whenever there is a change in the check sheet processing status, a value is saved to lpdwTransactionNumber, lpwMainStatus, and lpwSubStatus.

BiSCNMICRCancelStatusBack

Cancels the reading status notification request registered using either of BiSCNMICRSetStatusBackFunction.

Syntax

int **BiSCNMICRCancelStatusBack***(int nHandle)*

Argument

nHandle: Specifies the handle. This is an INT type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect

BiSetNumberOfDocuments

Specifies the number of docments to read using BiSCNMICRFunctionContinuously.

Syntax

int **BiSetNumberOfDocuments** *(int nHandle, BYTE bNumber)*

Argument

nHandle: Specifies the handle. This is an INT type.

bNumber: Specifies the number (0 to 100) of documents to read. This is a BYTE type. The initial value is 0.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-1110	ERR_LESS_CHECKS	The number of checks specified for BiSetNumberOfDocuments cannot be read.

Explanation

If 0 is specified for bNumber, all the documents set in the ASF (Auto Sheet Feeder) are read. If 1 or larger is specified for bNumber, reading ends when reading of the specified number of documents is complete. The setting made using this API is valid until BiCloseMonPrinter is run. If reading of the documents set in the ASF is completed before reading the number of documents specified using this API, the error code (ERR_LESS_CHECKS) is sent, indicating that the specified number of documents have not been read to SubStatus of the reading status MF_FUNCTION_DONE.

Note

The setting made using this API becomes valid when MF_PROCESS.bActivationMode of BiSCNMICRFunctionContinuously is set to MF_ACTIVATE_MODE_HIGH_SPEED. However, if MF_ACTIVATE_MODE_CONFIRMATION is set, the setting made using this API is not valid.

BiGetMicrText

Gets MICR text or OCR text.

Syntax

int **BiGetMicrText** *(int nHandle, DWORD dwTransactionNumber, LPMF_MICR ptMicr)*

Argument

nHandle: Specifies the handle. This is an INT type.

dwTransactionNumber: Specifies the transaction number (ID) for the MICR text acquired. This is a DWORD type.

ptMicr: Specifies the memory address of the MF_MICR structure. This is an LPMF_MICR type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-140	ERR_BUFFER_OVER_FLOW	Buffer overflow error
-220	ERR_NOT_FOUND	No data error
-440	ERR_MICR	Printer failed in MICR reading
-470	ERR_NOT_EXEC	Process not being executed
-1040	ERR_MICR_NODATA	MICR data is not existing
-1050	ERR_MICR_BADDATA	MICR data is not able to recognize
-1060	ERR_MICR_PARSE	MICR data can not be parsed
-1070	ERR_MICR_NOISE	Noise error has occurred during MICR reading

Explanation

Gets the MICR text read with BiSCNMICRFunctionContinuously. After the reading status MF_DATARECEIVE_DONE notification, the reading results are saved in the various parameters of the MF_MICR structure by specifying the relevant transaction number (ID) in dwTransactionNumber. When getting the MICR reading result, specify MF_MICR_USE_MICR in bMicOcrSelect of the MF_MICR structure, when getting the OCR reading result, specify MF_MICR_USE_OCR, and when getting the logical multiplication of the MICR and OCR reading results, specify MF_MICR_USE_MICR | MF_MICR_USE_OCR.
The MICR/OCR reading result is set to szMicrStr of MF_MICR structure.

Information on reliability of the read characters is set to stOcrReliableInfo of MF_MICR structure.
Specifying MF_MICR_USE_MICR | MF_MICR_USE_OCR for the bMicOcrSelect makes a comparison between the reading result of MICR and that of OCR, and if any difference is found, "?" will be returned.

Note

- ❏ The interval during which the MICR reading result and OCR reading result corresponding to the transaction number (ID) can be acquired is from MF_DATARECEIVE_DONE notification to MF_CHECKPAPER_PROCESS_DONE notification.
  When the process is returned from the MF_CHECKPAPER_PROCESS_DONE notification handler to the API, the MICR reading result/OCR reading result saved by the API is discarded.

- ❏ MICR magnetic waveform data is sent from the device when starting the reading. The position information is acquired from the magnetic waveform data. Therefore, if the position information could not be acquired from the magnetic waveform data when MF_MICR_USE_OCR has been specified to bMicOcrSelect, the OCR recognition cannot be made.

- ❏ When the font to read MICR (MF_MICR.bFont) is CMC7, the OCR recognition result (MF_OCR_AB.stOcrReliableInfo) cannot be obtained. The relation between bMicOcrSelect and bFont in MF_MICR structure for OCR reading process is described below.

bFont	**bMicOcrSelect**		
	MF_MICR_USE_MICR	**MF_MICR_USE_OCR**	**MF_MICR_USE_MICR | MF_MICR_USE_OCR**
MF_MICR_FONT_E13B	MICR magnetic waveform + OCR recognition	OCR recognition	MICR magnetic waveform + OCR recognition
MF_MICR_FONT_CMC7	MICR magnetic waveform	Error (ERR_MICR_NODATA)	Error (ERR_MICR_NODATA)

- ❏ Parsing can be used when the font to read MICR (MF_MICR.bFont) is E13B. When CMC7 is used, after calling BiGetMicrText, ERR_MICR_PARSE is returned as a return value.

BiMICRClearSpaces

Clears spaces included in MICR data obtained using BiGetMicrText.

Syntax

int **BiMICRClearSpaces** *(int nHandle, BYTE bClearSpace)*

Argument

nHandle: Specifies the handle. This is an INT type.

bClearSpace: Specifies clearance of spaces in MICR data. Specify the following constants. The default value is CLEAR_SPACE_DISABLE. This is a BYTE type.

Constant	Description
CLEAR_SPACE_DISABLE	Does not clear spaces.
CLEAR_SPACE_ENABLE	Clears spaces.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed

Explanation

The MICR data to be cleared is the following members in MF_MICR structure.

Member	Description
szMicrStr	MICR character string
stOcrReliableInfo	Information on reliability of MICR character recognition

The above settings are valid from the point when BiGetMicrText is invoked after invoking this API until BiCloseMonPrinter is invoked.

BiSetOcrABAreaOrigin

Specifies the origin of the OCR area for the MF_OCR_AB structure.

Syntax

int **BiSetOcrABAreaOrigin** *(int nHandle, BYTE bOrigin)*

Argument

nHandle: Specifies the handle. This is an INT type.

bOrigin: Specifies the origin of the OCR area. Specify the following values. The default value is OCR_ORIGIN_TOP_LEFT. This is a BYTE type.

Constant	Description
OCR_ORIGIN_TOP_LEFT	Sets the origin to the top left corner in relation to the document insertion direction.
OCR_ORIGIN_BOTTOM_LEFT	Sets the origin to the bottom left corner in relation to the document insertion point.
OCR_ORIGIN_TOP_RIGHT	Sets the origin to the top right corner in relation to the document insertion point.
OCR_ORIGIN_BOTTOM_RIGHT	Sets the origin to the bottom right corner in relation to the document insertion point.

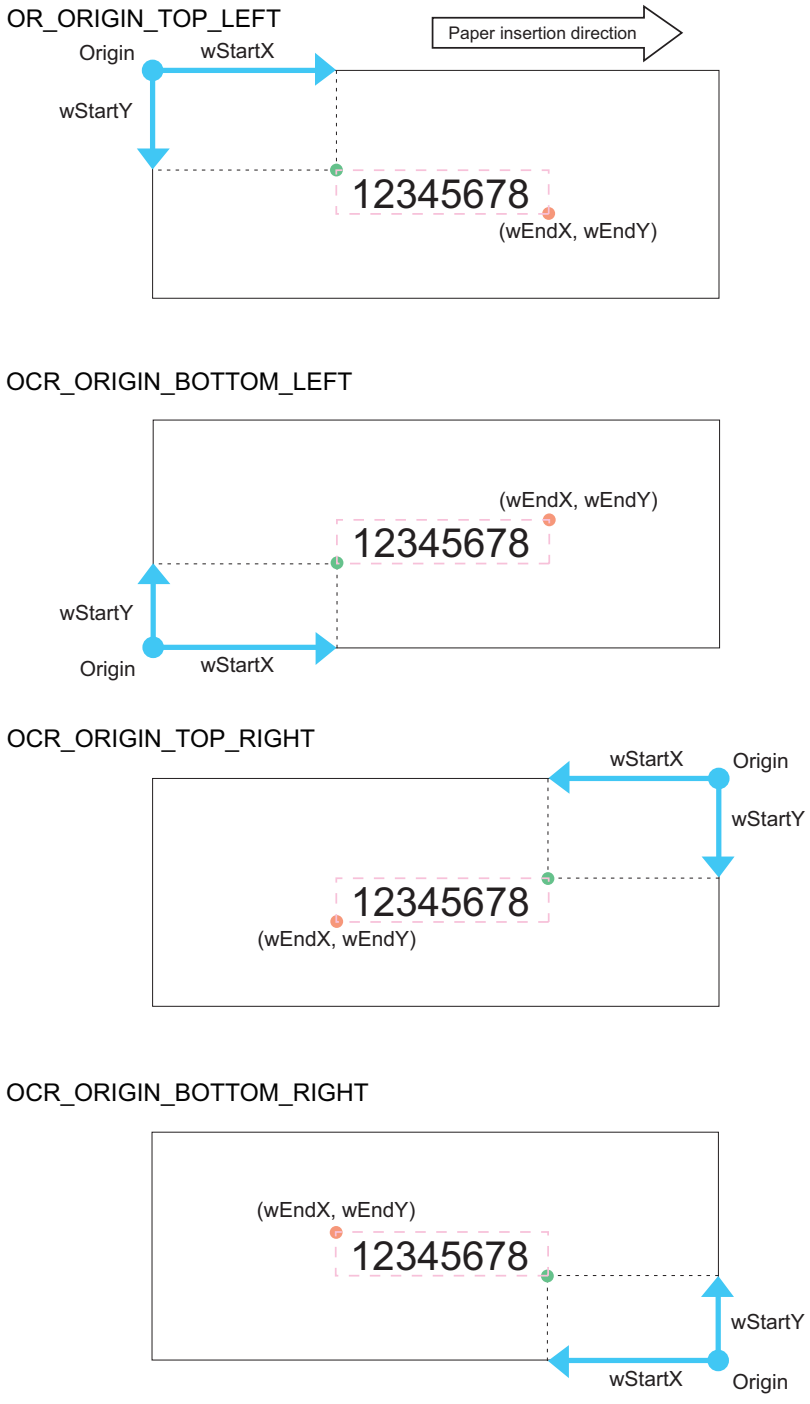
Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed

Explanation

The original specified using this API is valid until BiCloseMonPrinter is invoked.

See the following illustration for defining ranges with different origins.



BiGetOcrABText

Performs the OCR recognition for the OCR-A font or the OCR-B font and acquires the result.

Note

This API cannot be used when an image editing software is used to edit the image created by the driver.

Syntax

int **BiGetOcrABText**
(int nHandle, DWORD dwTransactionNumber, BYTE bImageSource, LPCSTR szFileName, LPMF_OCR_AB ptOcrAB)

Argument

nHandle: Specifies the handle. This is an INT type.

dwTransactionNumber: Specify a transaction number (ID) targeted for the OCR recognition. This is a DWORD type.

bImageSource: Specify an image targeted for the OCR recognition. One of the following values can be specified. This is a BYTE type.

Constant	Description
OCR_SOURCE_TRANSACTION_NUMBER	For an image file stored in the driver.
OCR_SOURCE_IMAGE_FILE	For an image file saved by the driver.

szFileName: Specify an image file name targeted for the OCR recognition. This is an LPCSTR type.

ptOcrAB: Specify the memory address of the MF_OCR_AB structure. For the MF_OCR_AB structure, refer to "MF_OCR_AB" on page 4-162. This is an LPMF_OCR_AB type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-220	ERR_NOT_FOUND	No data error
-230	ERR_IMAGE_FILEOPEN	Open failure
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-1040	ERR_MICR_NODATA	MICR data is not existing

Explanation

Performs the OCR recognition for the OCR-A font or the OCR-B font and acquires the result. Necessary conditions for the OCR recognition other than the targeted images are set to the MF_OCR_AB structure. The OCR recognition results are stored in the OUT attribute parameter of the MF_OCR_AB structure.

- ❏ When OCR_SOURCE_TRANSACTION_NUMBER is specified to bImageSource, images that are read immediately before and stored in the driver can be targeted for the OCR recognition. In this case, szFileName is ignored. After MF_DATARECEIVE_DONE is notified, the recognition result is stored in the OUT attribute parameter of the MF_OCR_AB structure by specifying the pertinent transaction number to dwTransactionNumber.

- ❏ When OCR_SOURCE_IMAGE_FILE is specified to bImageSource, the image files saved by the driver are targeted for OCR recognition. In this case, dwTransactionNumber is ignored.

Whenever image files exist it is OK to execute this API. The image files must meet the following conditions.

- ❏ Saved when either EPS_BI_SCN_BITMAP or EPS_BI_SCN_TIFF256 is specified with BiSCNSetImageFormat.

- ❏ Saved when EPS_BI_SCN_8BIT is specified to bColorDept parameter with bBiSCNSetImageQuality.

- ❏ Saved when EPS_BI_SCN_MANUAL is specified to bExOption parameter with BiSCNSetImageQuality.

Note

When OCR_SOURCE_TRANSACTION_NUMBER is specified to bImageSource, the period that the OCR recognition result for the transaction number (ID) can be acquired is from when MF_DATARECEIVE_DONE is notified to when MF_CHECKPAPER_PROCESS_DONE is notified.

The OCR recognition results that TM-S1000 API has acquired are discarded when the process is returned to TM-S1000 API from the MF_CHECKPAPER_PROCESS_DONE notification handler.

BiGetScanImage

Gets the scan image.

Syntax

int **BiGetScanImage***(int nHandle, DWORD dwTransactionNumber, LPMF_SCAN ptScan)*

Argument

nHandle: Specifies the handle. This is an INT type.

dwTransactionNumber: Specifies the transaction number (ID) for the scan image acquired. This is a DWORD type.

ptScan: Specifies the memory address of the MF_SCAN structure. This is an LPMF_SCAN type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-70	ERR_TIMEOUT	A time out error occurred
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-220	ERR_NOT_FOUND	No data error
-450	ERR_SCAN	Printer failed in image scanning
-470	ERR_NOT_EXEC	Process not being executed
-1080	ERR_SCN_COMPRESS	Scan image data compressing error

Explanation

Gets the scan image scanned with BiSCNMICRFunctionContinuously / BiSCNMICRFunctionPostPrint. After the scanning status MF_DATARECEIVE_DONE notification, the scanning results are saved in the various parameters of the MF_SCAN structure by specifying the relevant transaction number (ID) in dwTransactionNumber. When configuring a valid value for sResolution, bAddInfoDataSize and pAddInfoData that are part of MF_SCAN structure, the image which these values refer to will be stored. To get the front side scanning results, specify MF_SCAN_FACE_FRONT with BiSCNSelectScanFace, then execute this API. To get the back side scanning results, specify MF_SCAN_FACE_BACK with BiSCNSelectScanFace, then execute this API.

Note

- The BiSCNSetImageQuality, BiSCNSetImageFormat, and BiSCNScanArea setting values are reflected in the scanning result when BiSCNMICRFunctionContinuously is executed. Set BiSCNImageQuality, and BiSCNSetImageFormat before executing BiSCNMICRFunctionContinuously scanning.

- When this API is executed, the scanned image address is set in lpvScanData of the MF_SCAN structure. This memory is obtained automatically by the API, and it must not be discarded. Therefore, it is necessary for the application to discard it at the appropriate time. To discard this memory, use the application to specify the address of this memory in the GlobalFree function of the WindowsAPI.

- Be sure to configure a valid value for bAddInfoDataSize and pAddInfoData that are part of MF_SCAN structure.

- The interval during which the scan image corresponding to the transaction number (ID) can be acquired is from MF_DATARECEIVE_DONE notification to MF_CHECKPAPER_PROCESS_DONE notification. When the process is returned from the MF_CHECKPAPER_PROCESS_DONE notification handler to the API, the scan image saved by the API is discarded.

BiGetBarcodeData

Decodes a barcode from the scanned image data, and obtains the result.

Syntax

int **BiGetBarcodeData** *(int nHandle, DWORD dwTransactionNumber, LPMF_BARCODE ptBarcode)*

Argument

nHandle: Specifies the handle. This is an INT type.

dwTransactionNumber: Specifies a transaction number (ID) for the BARCODE decode. This is a DWORD type.

ptBarcode: Specifies the memory address of the MF_BARCODE structure. This is an LPMF_BARCODE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-220	ERR_NOT_FOUND	No data error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-1130	ERR_BARCODE_NODATA	Barcode cannot be detected.

Explanation

Decodes a barcode from the scanned image data, and obtains the result.
The result of barcode decode is set in the MF_BARCODE structure. For the details, refer to "MF_BARCODE" on page 4-176.
To decode a barcode, specify the front/back side with BiSCNSelectScanFace.

Note

- The decoding result of BARCODE corresponding to the transaction number (ID) is obtainable from the processing status of MF_DATARECEIVE_DONE to the notification of MF_CHECKPAPER_PROCESS_DONE. The decoding result of BARCODE kept by Status API is discarded when the handler of MF_CHECKPAPER_PROCESS_DONE sends back the processing to Status API.
- When decoding multiple barcodes at the same time, the correct result cannot be obtained if the barcodes other than given below are decoded.
  - Barcode symbols are the same
  - Barcodes are lined in the horizontal direction

- This API is MF_SET_BARCODE_FRONT_PARAM or MF_SET_BARCODE_BACK_PARAM, and can obtain the decoding result different from the MF_BARCODE structure which was set earlier. In such a case, where to eject or the flanking process is set according to the current setting. The decoding result can be obtained even if the MF_BARCODE structure is not set earlier.

BiDecodeBarcode

Decodes a barcode from the specified image file, and obtains the result.

Syntax

int **BiDecodeBarcode** *(int nHandle, LPCSTR szFileName, LPMF_BARCODE ptBarcode)*

Argument

nHandle: Specifies the handle. This is an INT type.

szFileName: Specifies the name of the image file (file pass). This is a LPCSTR type.

ptBarcode: Specifies the memory address of the MF_BARCODE structure. This is an LPMF_BARCODE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-230	ERR_IMAGE_FILEOPEN	Open failure
-240	ERR_IMAGE_UNKNOWNFORMAT	Format injustice
-1130	ERR_BARCODE_NODATA	Barcode cannot be detected.

Explanation

Decodes a barcode from the image file, and obtains the result.
The result of barcode decode is set in the MF_BARCODE structure. For the details, refer to "MF_BARCODE" on page 4-176.

Note

- ❏ Make sure to use the image file created in the following way.
  - Saved file with EPS_BI_SCN_BITMAP or EPS_BI_SCN_TIFF256 specified in BiSCNSetImageFormat
  - Saved file with EPS_BI_SCN_8BIT to bColorDepth specified in BiSCNSetImageQuality
  - Saved file with EPS_BI_SCN_SHARP to bExOption specified in BiSCNSetImageQuality
- ❏ When decoding multiple barcodes at the same time, the correct result cannot be obtained if the barcodes other than given below are decoded.
  - Barcode symbols are the same
  - Barcodes are lined in the horizontal direction

BiDecodeBarcodeMemory

Decodes the barcode from the image data on the application memory, and obtains the result.

Syntax

int **BiDecodeBarcodeMemory** *(int nHandle, LPBYTE lpImageBuffer, DWORD dwBufferSize, LPMF_BARCODE ptBarcode)*

Argument

nHandle:	Specifies the handle. This is an INT type.
lpImageBuffer:	Specifies the memory of image data. This is a LPBYTE type.
dwBufferSize:	Specifies the memory size of specified image data in *lpImageBuffer*. This is a DWORD type.
ptBarcode:	Specifies the memory address of the MF_BARCODE structure. This is an LPMF_BARCODE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-240	ERR_IMAGE_UNKNOWNFORMAT	Format injustice
-1130	ERR_BARCODE_NODATA	Barcode cannot be detected.

Explanation

The result of barcode decode is set in the MF_BARCODE structure. For the details, refer to "MF_BARCODE" on page 4-176.

Note

- ❏ Make sure to use the image data which was created as following ways.
  - Created data with EPS_BI_SCN_BITMAP or EPS_BI_SCN_TIFF256 specified in BiSCNSetImageFormat
  - Created data with EPS_BI_SCN_8BIT to bColorDepth specified in BiSCNSetImageQuality
  - Created data with EPS_BI_SCN_SHARP to bExOption specified in BiSCNSetImageQuality
- ❏ When decoding multiple barcodes at the same time, the correct result cannot be obtained if the barcodes other than given below are decoded.
  - Barcode symbols are the same
  - Barcodes are lined in the horizontal direction

BiGetTransactionNumber

Gets the currently set transaction number.

Syntax

int **BiGetTransactionNumber** *(int nHandle, LPDWORD lpdwTransactionNumber)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpdwTransactionNumber: Specifies the memory address where the transaction number (ID) is saved. This is an LPDWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Note

- The transaction number (ID) acquired by this API is the value used for the next scanning process, and it cannot be used as a parameter for BiGetMicrText and BiGetScanImage.
- When getting MICR text or a scan image with BiGetMicrText and BiGetScanImage, use the transaction number (ID) provided with the scanning status MF_DATARECEIVE_DONE or MF_CHECKPAPER_PROCESS_DONE.

BiSetTransactionNumber

Sets the transaction number.

Syntax

int **BiSetTransactionNumber** *(int nHandle, DWORD dwTransactionNumber)*

Argument

nHandle: Specifies the handle. This is an INT type.

dwTransactionNumber: Specifies the transaction number (ID) to set. The range of values that can be set is from 0 to 999999999. This is a DWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Sets the transaction number (ID) used for sequential printing, BiSCNMICRFunctionContinuously scanning status notification. After MF_CHECKPAPER_PROCESS_START notification with BiSCNMICRFunctionContinuously, 1 is added to the setting value. If MF_CHECKPAPER_PROCESS_START notification is performed at the maximum value of 999999999, the setting value returns to 0.

The transaction number (ID) default value is set to 1.

<About sequential printing function>
It is the function that specifies a format for transaction number (ID) printing. It prints using the format of the keyword surrounded by <>. If the keyword surrounded by <> is specified to the endorse printing character string before the reading process, character string is made using the value acquired by BiGetTransactionNumber. The character string is valid until either BiBufferedPrint MF_PRT_CLEAR or BiBufferedPrint invokes BiPrintText in the MF_PRT_EXEC status.
If the keyword surrounded by <> is specified to the endorse printing character string during the MF_DATARECEIVE_DONE callback, character string is made using the transaction ID for the check paper. This character string is cleared after the MF_DATARECEIVE_DONE callback is notified.

There are three patterns for the keywords that can be specified for the sequential printing as follows:
<0000>: The number of "0"s indicates the number of columns to be printed. The number of columns that can be set is from 1 to 9. If the transaction number set is less than the number of the columns, 0 is added automatically.

<xxxx>: The number of "x"s indicates the number of columns to be printed. The number of the columns that can be set is from 1 to 9. If the transaction number set is less than the number of the columns, a space is added automatically.

<llll>: (small letter of "L" in one-byte): The number of "l"s indicates the number of columns to be printed. The number of the columns that can be set is from 1 to 9. If the transaction number set is less than the number of the columns, the columns are left aligned automatically.

* "<" and ">" are used as special symbols. When printing "<" or ">", specify &< or &> respectively.

* If a keyword is specified that is outside the rule mentioned above, for example <00xxabc> (0 and x are mixed, the characters other than 0 or x are included), the character string surrounded by "<" and ">" is printed as a usual character string.

Note

With sequential printing, if a number of digits n smaller than the transaction number (ID) currently set is specified, the latter n digits of the transaction number (ID) are actually printed.

Example:
When the transaction number (ID) currently set is 12345 and the sequential printing is specified with 4-column <xxxx>, “2345” is actually printed.

BiGetPrintStation

Gets the set station for printing.

Syntax

int **BiGetPrintStation** *(int nHandle, LPWORD lpwStation)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpwStation: Specifies the memory address where the station for printing is saved. This is an LPWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

BiSetPrintStation

Sets the station for printing.

Syntax

int **BiSetPrintStation** *(int nHandle, WORD wStation)*

Argument

nHandle: Specifies the handle. This is an INT type.

wStation: Specifies the station for printing. One of the following values can be set. This is a WORD type.

Constant	Description
MF_ST_E_ENDORSEMENT	Sets electronic endorsement as the station for printing.
MF_ST_E_ENDORSEMENT_BACK	Same as MF_ST_E_ENDORSEMENT.
MF_ST_E_ENDORSEMENT_FRONT	Sets electronic endorsement to front side as the station for printing.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Specifies the BiPrintText, and BiPrintImage stations for printing.

BiPrintText

Executes test printing.

Syntax

int **BiPrintText** *(int nHandle, LPSTR szText, MF_DECORATE tDecorate)*

Argument

nHandle: Specifies the handle. This is an INT type.

szText: Specifies the memory address where the text for printing is saved. This is an LPSTR type.

tDecorate: Specifies the DECORATE structure where the text decoration information is saved. This is a MF_DECORATE type.

```
DECORATE structure
struct {
DWORD     dwAttribute;
WORD      wFont;
LPSTR     szFontName;
WORD      wFontSize;
} MF_DECORATE, *LPMF_DECORATE;
```

dwAttribute: Specifies the text attributes. The following values can be set individually or in multiples. If no attribute is specified, specify MF_PRINT_NO_ATTRIBUTE. This is a DWORD type.

Constant	Description
MF_PRINT_BOLD	Prints in bold
MF_PRINT_UNDERLINE_1	Adds 1-dot wide underlining
MF_PRINT_UNDERLINE_2	Adds 2-dot wide underlining
MF_PRINT_REVERSEVIDEO	Prints reversed
MF_PRINT_BLACK	Prints in the 1st color (normally black) (MF_PRINT_1ST_COLOR is the same)
MF_PRINT_COLOR	Prints in the 2nd color (MF_PRINT_2ND_COLOR is the same)
MF_PRINT_MIXED	Prints characters in a mixture of the 1st and 2nd color.

wFont: Specifies the type of font. The selectable value is as follows. This is a WORD type.

Constant	Description
MF_PRINT_SYSTEMFONT	Prints with the system font

szFontName: When MF_PRINT_SYSTEMFONT is specified with wFont, any system font can be used for printing by specifying the font name in this parameter. If the specified font is not present, printing is performed using the system default font. This is an LPSTR type.

wFontSize: Specifies the font size. When MF_PRINT_FONT_A / MF_PRINT_FONT_B is specified in wFont, one of the following values can be set. When MF_PRINT_SYSTEMFONT is specified in wFont, point units can be specified. Many system fonts support 8 to 72 pt. If the specified size does not agree, printing is performed using the nearest size font. Small sized characters are hard to read at 100/120 dpi. Arial font, size 15 point or larger is recommended.This is a WORD type.

Constant	Description
MF_PRINT_FONT_W1_H1	Prints at width x1 and height x1 size.
MF_PRINT_FONT_W1_H2	Prints at width x1 and height x2 (double height) size.
MF_PRINT_FONT_W2_H1	Prints at width x2 and height x1 (double width) size.
MF_PRINT_FONT_W2_H2	Prints at width x2 and height x2 (double height and width) size.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-80	ERR_ACCESS	Reading/writing with the device is not possible (printing in progress)
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed

Explanation

Prints text with the attributes specified with the MF_DECORATE structure, from the station specified in BiSetPrintStation.

Note

- When MF_PRINT_SYSTEMFONT is specified to wFont of the MF_DECORATE structure, the thickness of an underline depends on the font size. Therefore, even if MF_PRINT_UNDERLINE_1 or MF_PRINT_UNDERLINE_2 is specified, the thickness of the underlines is the same.
- Even if MF_PRINT_COLOR, MF_PRINT_MIXED is specified to dwAttribute of the MF_DECORATE structure, the character string printed is MF_PRINT_BLACK.

BiPrintImage

Executes image printing from a file.

Syntax

int **BiPrintImage** (*int nHandle, LPSTR pFileName*)

Argument

nHandle: Specifies the handle. This is an INT type.

pFileName: Specifies the location of the image file to print with a full path. This is an LPSTR type. Image file that can be specified are below.

- BMP format (Uncompressed image data only)
- JPEG format (Baseline DCT, Progressive)
- TIFF format
  (CCITT Group 3/Group 4 compressed data, uncompressed data only)

The following formats are not supported.

- BMP format (Compressed image data)
- JPEG format (Lossless compression, hierarchical coding)
- TIFF format (Palette color, PackBits/JPEG/LZW/ZIP compression)

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-230	ERR_IMAGE_FILEOPEN	Open failure
-240	ERR_IMAGE_UNKNOWNFORMAT	Format injustice
-1000	ERR_SIZE	Size excess error

Explanation

Prints images from the station specified in BiSetPrintStation.

Prints after enlarging or reducing in accordance with the print size specified in BiSetPrintSize.

NOTE

- If the image file specified with pFileName does not exist, the ERR_IMAGE_FILEOPEN error is returned. Furthermore, if the file specified in pFileName does not meet the rule, the ERR_IMAGE_UNKNOWNFORMAT error is returned.
- If the image size exceeds 4096*4096, an error of ERR_IMAGE_UNKNOWNFORMAT is returned even when the image format meets the supported format.

BiPrintMemoryImage

Executes image printing from memory.

Syntax

int **BiPrintMemoryImage***(int nHandle, LPBYTE lpbImageData, DWORD dwDataSize)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpbImageData: Specifies image data for printing. Image files that can be specified are below. This is an LPBYTE type.

- BMP format (Uncompressed image data only)
- JPEG format (Baseline DCT, Progressive)
- TIFF format
  (CCITT Group 3/Group 4 compressed data, uncompressed data only)

The following formats are not supported.

- BMP format (Compressed image data)
- JPEG format (Lossless compression, hierarchical coding)
- TIFF format (Palette color, PackBits/JPEG/LZW/ZIP compression)

dwDataSize: Specifies a size of image data to be printed. This is a DWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-80	ERR_ACCESS	Reading/writing with the device is not possible
-90	ERR_PARAM	Parameter error
-240	ERR_IMAGE_UNKNOWNFORMAT	Format injustice
-1000	ERR_SIZE	Size excess error

Explanation

Executes image printing for the station specified with BiSetPrintStation.
Prints after enlarging or reducing in accordance with the print size specified with BiSetPrintSize.

NOTE

- If file data that does not meet the rule is specified to lpbImageData, the ERR_IMAGE_UNKNOWNFORMAT error is returned.
- If the image size exceeds 4096*4096, an error of ERR_IMAGE_UNKNOWNFORMAT is returned even when the image format meets the supported format.

BiGetPrintSize

Gets the print size set in the station specified in BiSetPrintStation.

Syntax

int **BiGetPrintSize** *(int nHandle, LPWORD lpwWidth, LPWORD lpwHeight)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpwWidth: Specifies the memory address where the horizontal print size is saved. This is an LPWORD type.

lpwHeight: Specifies the memory address where the vertical print size is saved. This is an LPWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

BiSetPrintSize

Sets the print size.

Syntax

int **BiSetPrintSize** *(int nHandle, WORD wWidth, WORD wHeight)*

Argument

nHandle: Specifies the handle. This is an INT type.

wWidth: Horizontal print size (unit: mm). This is a WORD type.

wHeight: Vertical print size (unit: mm). This is a WORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-1000	ERR_SIZE	Size excess error

Explanation

Sets the size of the image printed with BiPrintImage,BiPrintMemoryImage. It enlarges or reduces it to fit in the specified size. The setting values are saved in each station that can be specified with BiSetPrintStation. If the size specified with this API exceeds the printing area of the station set with BiSetPrintStation, the ERR_ SIZE error is returned and the setting value is changed. The setting values specified in this API are saved until BiCloseMonPrinter is executed. The default value for horizontal print size and vertical print size is 0.

BiGetPrintPosition

Gets the currently set printing start position.

Syntax

int **BiGetPrintPosition** *(int nHandle, LPWORD lpwHorizontal, LPWORD lpwVertical)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpwHorizontal: Specifies the memory address where the horizontal printing start position is saved. This is an LPWORD type.

lpwVertical: Specifies the memory address where the vertical printing start position is saved. This is an LPWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Gets the setting value of the currently set printing start position. This API can only be executed if MF_ST_E_ENDORSEMENT is specified in BiSetPrintStation. This API can be called when Electronic Endorsement (MF_ST_E_ENDORSEMENT, MF_ST_E_ENDORSEMENT_BACK, MF_ST_E_ENDORSEMENT_FRONT) is specified for BiSetPrintStation.

BiSetPrintPosition

Sets the printing start position.

Syntax

int **BiSetPrintPosition** *(int nHandle, WORD wHorizontal, WORD wVertical)*

Argument

nHandle: Specifies the handle. This is an INT type.

wHorizontal: Horizontal printing start position (unit: mm). This is a WORD type.

wVertical: Vertical printing start position (unit: mm). This is a WORD type.

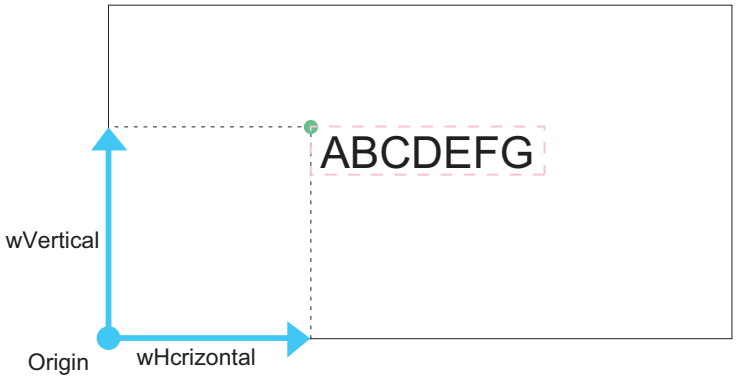
Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect

Explanation

Specifies the printing start position. This API can only be executed if MF_ST_E_ENDORSEMENT is specified in BiSetPrintStation. This API can be called when Electronic Endorsement (MF_ST_E_ENDORSEMENT, MF_ST_E_ENDORSEMENT_BACK, MF_ST_E_ENDORSEMENT_FRONT) is specified for BiSetPrintStation.
The printing start position specified in this API is saved until BiCloseMonPrinter is executed.

The origin for the printing start position specified in this API is the bottom left of the scan image of the back side of the check paper.
Refer to the model diagram below for the specified printing start position and print data expansion method.



Note

When multiple print data is expanded with the same printing start position, it is expanded in duplicate. Adjustment of the printing start position should be performed by the application.

BiSetEndorseDirection

Specifies the direction of E-Endorse.

Syntax

int **BiSetEndorseDirection** *(int nHandle, BYTE bDirection)*

Argument

nHandle: Specifies the handle. This is an INT type.

bDirection: Specifies the direction for E-endorse. One of the following values can be set. This is a BYTE type.

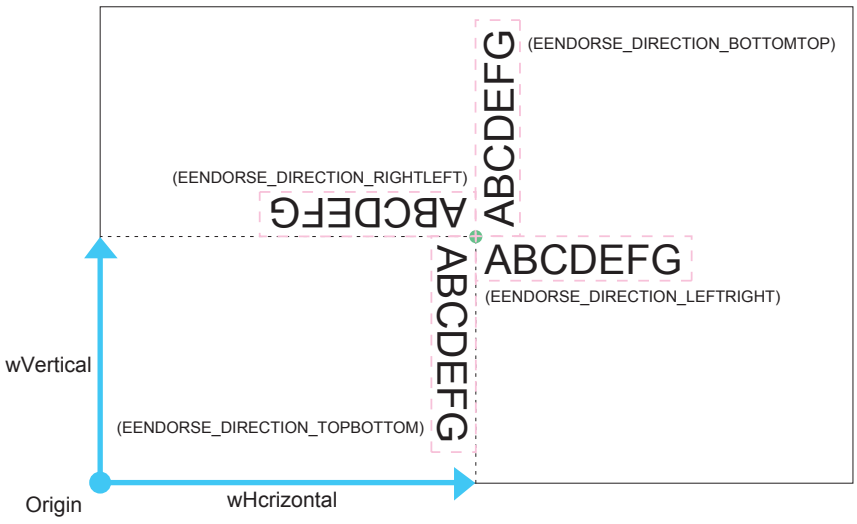
Constant	Value	Description
EENDORSE_DIRECTION_LEFTRIGHT	1	From left to right (normal direction)
EENDORSE_DIRECTION_TOPBOTTOM	2	From top to bottom (Rotate 90° clockwise)
EENDORSE_DIRECTION_RIGHTLEFT	3	From right to left (upside down)
EENDORSE_DIRECTION_BOTTOMTOP	4	From bottom to top (Rotate 90° counterclockwise)

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error

Explanation

The electric endorse printing can be executed by executing BiPrintText, BiPrintImage, BiPrintMemoryImage.
Selection of electric endorsement on the front side/backside can be made by setting BiSetPrintStation; however, the setting of this API is applied to both sides regardless of the front side/backside setting.
The electric endorsement is rotated around the supporting point specified with BiSetPrintPosition.



BiUpdateEndorseText

Updates an endorse character string.

Syntax

int **BiUpdateEndorseText***(int nHandle, LPSTR lpString[3], DWORD, dwAttribute[3], WORD wFont[3], WORD wFontSize[3])*

Argument

nHandle: Specifies the handle. This is an INT type.

lpString[3]: Specifies an endorse character string. lpString[0] is the first line, lpString[1] is the second line, and lpString[2] is the third line. This is an LPSTR type.

dwAttribute[3]: Specifies an attribute of the endorse character string. dwAttribute[0] is the first line, dwAttribute[1] is the second line, and dwAttribute[2] is the third line. This is a DWORD type.

wFont[3]: Specifies a font of the endorse character string. wFont[0] is the first line, wFont[1] is the second line, and wFont[2] is the third line. This is a WORD type.

wFontSize[3]: Specifies a font size of the endorse character string. wFontSize [0] is the first line, wFontSize [1] is the second line, and wFontSize [2] is the third line. This is a WORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The endorse character string can be updated by invoking this API from the notification handler of MF_DATARECEIVE_DONE, the reading status of BiSCNMICRFunctionPostPrint. When the MF_PRINT structure is not set with BiSCNMICRFunctionPostPrint and endorse printing is not executed, if the API is executed, the ERR_EXEC_FUNCTION error is returned and the endorse character string cannot be updated.

BiBufferedPrint

Switches to the buffered print mode and executes buffered printing.

Syntax

int **BiBufferedPrint** *(int nHandle, DWORD dwFunction)*

Argument

nHandle: Specifies the handle. This is an INT type.

dwFunction: Specifies the function to execute. One of the following values can be set. This is a DWORD type.

Constant	Description
MF_PRT_BUFFERING	Buffers the print data.
MF_PRT_EXEC	Prints the buffered print data.
MF_PRT_CLEAR	Clear the buffered print data and exits buffering mode.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-80	ERR_ACCESS	Cannot read/write
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Provides data buffering and prints buffered data to provide buffered printing. The execution status of this API is saved in each station that can be specified with BiSetPrintStation. It is possible to improve printing performance using buffered printing using this API as compared with executing the printing methods individually (BiPrintText, BiPrintImage,BiPrintMemoryImage).

Furthermore, it is possible to expand the electronic endorsement while receiving the scan data when the electric endorsement (MF_ST_E_ENDORSEMENT, MF_ST_E_ENDORSEMENT_BACK, MF_ST_E_ENDORSEMENT_FRONT) is specified in BiSetPrintStation by invoking this API, buffering the print data and performing buffered printing. By first expanding data that does not need to be changed using this API to electronic endorsement, it is possible to limit data expanded in the scanning status MF_DATARECEIVE_DONE notification to the data that requires occasional updating.

BiSetTransactionNumberWithIncremental

Sets the transaction number and the incremental value.

Syntax

int **BiSetTransactionNumberWithIncremental**
(int nHandle, DWORD dwTransactionNumber, DWORD dwIncremental)

Argument

nHandle: Specifies the handle. This is an INT type.

dwTransactionNumber: Specifies the transaction number (ID) to set. The range of values that can be set is from 0 to 999999999. This is a DWORD type.

dwIncremental: Specifies the incremental value of transaction number to set. The range of values that can be set is from 1 to 999999999.This is a DWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-80	ERR_ACCESS	Cannot read/write
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Sets the transaction number (ID) used for sequential printing, BiSCNMICRFunctionContinuously scanning status notification. The set value will be incremented by dwIncremental after the MF_CHECKPAPER_PROCESS_START notification with BiSCNMICRFunctionContinuously and BiSCNMICRFunctionPostPrint. If MF_CHECKPAPER_PROCESS_START notification is performed at the maximum value of 999999999, the setting value returns to 0. The transaction number (ID) default value is set to 1. The default incremental value is set to 1.

<About sequential printing function>
It is the function that specifies a format for transaction number (ID) printing. It prints using the format of the keyword surrounded by <>. If the keyword surrounded by <> is specified to the endorse printing character string before the reading process, a character string using the value acquired by BiGetTransactionNumber is made. The character string is valid until either BiBufferedPrint MF_PRT_CLEAR or BiBufferedPrint invokes BiPrintText in the of MF_PRT_EXEC status.

If the keyword surrounded by <> is specified to the endorse printing character string during the MF_DATARECEIVE_DONE callback, a character string using the transaction ID for the check paper is made. This character string is cleared after the MF_DATARECEIVE_DONE callback is notified.

There are three patterns for the keywords that can be specified for the sequential printing as follows:

<0000>: The number of "0"s indicates the number of columns to be printed. The number of columns that can be set is from 1 to 9. If the transaction number set is less than the number of columns, 0 is added automatically.

<xxxx>: The number of "x"s indicates the number of the columns to be printed. The number of the columns that can be set is from 1 to 9. If the transaction number set is less than the number of the columns, a space is added automatically.

<llll>(small letter of "L" in one-byte): The number of "l"s indicates the number of columns to be printed. The number of columns that can be set is from 1 to 9. If the transaction number set is less than the number of columns, the columns are left aligned automatically.

* "<" and ">" are used as special symbols. When printing "<" or ">", specify &< or &> respectively.

* If a keyword is specified that is outside the rule mentioned above, for example <00xxabc> (0 and x are mixed, the characters other than 0 or x are included), the character string surrounded by “<“ and “>“ is printed as a usual character string.

Note

With sequential printing, if a number of digits n smaller than the transaction number (ID) currently set is specified, the latter n digits of the transaction number (ID) are actually printed.

Example:
When the transaction number (ID) currently set is 12345 and the sequential printing is specified with 4-column <xxxx>, “2345” is actually printed.

BiSetBehaviorToScnResult

Sets the behavior to the result for scanning.

Syntax

int **BiSetBehaviorToScnResult**(*int nHandle, BYTE bEject, BYTE bStamp, BYTE bNextCheck)*

Argument

nHandle: Specifies the handle. This is an INT type.

bEject: Sets the ejection method of check papers. This is a BYTE type.

Constant	Description
MF_EJECT_MAIN_POCKET	Ejects into the main pocket.
MF_EJECT_SUB_POCKET	Ejects into the sub pocket.
MF_EJECT_NOEJECT	Does not eject.

bStamp: Sets whether to enable a franker. This is a BYTE type.

Constant	Description
MF_STAMP_DISABLE	Does not perform franking.
MF_STAMP_ENABLE	Performs franking.

bNextCheck: Sets whether to feed the next check paper when ejecting the current one. This is a BYTE type.

Constant	Description
MF_PROCESS_CONTINUE_OVERLAP	Starts the next reading process while ejecting documents.
MF_PROCESS_CONTINUE_NOOVERLAP	Starts the next reading process after ejecting documents.
MF_PROCESS_CONTINUE_CANCEL	Cancels the next reading process.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error

Explanation

It becomes possible to call it in the call back function, in the case that bActivationMode of MF_PROCESS structure is bound to MF_ACTIVATE_MODE_CONFIRMATION. In the case that designate MF_ACTIVATE_MODE_HIGH_SPEED and called it is disregarded.

BiSetPaperThickness

Specifies the double feed threshold.

Syntax

int **BiSetPaperThickness** *(int nHandle, WORD wThreshold)*

Argument

nHandle: Specifies the handle. This is an INT type.

wThreshold: Specifies the double feed threshold. The valid specification range is 1 to 40 (0.01 mm to 0.40 mm). This is a WORD type.
When "0" is specified, the default value is selected.
Paper thickness exceeding the specified value is detected as double feed.
The default value is listed below.

MfPaperType	Default value
MF_PAPER_TYPE_CHECK	0.17 mm
MF_PAPER_TYPE_OTHER	0.23 mm

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed

Explanation

The threshold set using this API ignores the thresholds defined for each paper type using the MF_PROCESS structure in advance (See "MF_PROCESS" on page 4-152) and applies to all the paper types.

The thresholds set using this API are valid until BiCloseMonPrinter is invoked.

BiRingBuzzer

Sounds a buzzer.

Syntax

int **BiRingBuzzer***(int nHandle, BYTE bTone, BYTE bCount, WORD wOnTime, WORD wOffTime)*

Argument

nHandle: Specifies the handle. This is an INT type.

bTone: Specifies the buzzer tone. This is a BYTE type.

Constant	Description
MF_BUZZER_TONE_HIGH	High-pitched sound
MF_BUZZER_TONE_MIDDLE	Middle-pitched sound
MF_BUZZER_TONE_LOW	Low-pitched sound

bCount: Specifies the number of buzzers. The valid setting value is 1 to 8. When a value exceeding 8 is set, it will be rounded to 8. This is a BYTE type.

wOnTime: Specifies the buzzer tone duration. The valid setting value is 100 to 800 (unit: mm). The specification must be made in 100 mm unit. A value less than 100 is rounded off to the nearest hundred. When a value exceeding 800 is set, it will be rounded to 800. This is a WORD type.

wOffTime: Specifies the off time of buzzer tone. The valid setting value is 100 to 800 (unit: mm). The setting must be made in 100 mm unit. A value less than 100 is rounded off to the nearest hundred. A value outside the valid range is rounded to the approximate value that can be specified.
0 (zero) can be set as well as valid values for the setting. When 0 is specified, the buzzer keeps sounding during the duration calculated by the following expression: Ring duration x Number of buzzers. No value is rounded to 0. This is a WORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error

Explanation

Setting all the parameters of *bTone*, *bCount*, *wOnTime* and *wOffTime* to 0 (zero) stops the buzzer.

This also applies to the buzzer specified by the MF_BASE structure.

BiSetWaterfallMode

Specifies the Waterfall mode.

Syntax

int **BiSetWaterfallMode** *(int nHandle, BYTE bWaterfallMode)*

Argument

nHandle: Specifies the handle. This is an INT type.

bWaterfallMode:
Specifies the Watefallmode. The default value is WATERFALL_MODE_DISABLE. This is a BYTE type.

Constant	Description
WATERFALL_MODE_DISABLE	Disables Waterfall mode.
WATERFALL_MODE_STANDARD	Enables Waterfall mode. Ejects to the main pocket when starting the reading process. When the main pocket's near full is detected, the ejection pocket is switched to the sub pocket. When the sub pocket's near full is detected, the ejection pocket is switched to the main pocket.
WATERFALL_MODE_INHERIT_POCKET	Enables Waterfall mode. Ejects to the ejection pocket of the previous reading process. (For example, the previous ejection pocket is the sub pocket, ejects to the sub pocket.) When a pocket near full is detected, the ejection pocket is switched to the other pocket. When a pocket near full has already been detected when starting the reading process, the ejection pocket is switched to the other pocket. The ejection pocket, however, is not switched when the other pocket's near full has been detected.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed

Explanation

When a pocket near full is detected when the Waterfall mode is started, the first document is ejected to the following pocket.

Waterfall mode	Pocket status		Ejection pocket
	Main Pocket	Sub Pocket	
WATERFALL_MODE_STANDARD	Not NearFull	Not NearFull	Main Pocket
	NearFull	Not NearFull	Sub Pocket
	Not NearFull	NearFull	Main Pocket
	NearFull	NearFull	Main Pocket
WATERFALL_MODE_INHERIT_POCKET	Not NearFull	Not NearFull	Inherit Pocket
	NearFull	Not NearFull	Sub Pocket
	Not NearFull	NearFull	Main Pocket
	NearFull	NearFull	Inherit Pocket

Note

- ❏ When a pocket near full is detected from both pockets, it is recommended to set NearFullSelect of MF_PROCESS structure to MF_NEARFULL_NOT_PERMIT, to stop the reading process. (See "MF_PROCESS" on page 4-152.)
- ❏ If the TM-S1000 is scanning, this API returns ERR_EXEC_FUNCTION. This API works for High speed and Confirmation. In the case of Confirmation, this API works for initial-value of BiSetBehaviorToScnResult.
- ❏ Ejection pocket setting for each error of MF_PROCESS structure is ignored.

BiGetIQAResult

Gets IQA result.

Syntax

int **BiGetIQAResult** *(int nHandle, DWORD dwTransactionNumber , LPMF_IQA_RESULT lpResult)*

Argument

nHandle: Specifies the handle. This is an INT type.

dwTransactionNumber:
Specifies the transaction number (ID) for the MICR text acquired. This is a DWORD type.

lpResult: Specifies the memory address of the MF_IQA_RESULT structure. This is an LPMF_IQA_RESULT type. The following result is set for each IQA validation item.

Constant	Description
IQARESULT_NOT_TESTED	IQA validation is not executed.
IQARESULT_PASS	Passed IQA validation.
IQARESULT_NOT_PASS	Not passed IQA validation.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-140	ERR_BUFFER_OVER_FLOW	Buffer overflow error
-220	ERR_NOT_FOUND	No data error
-470	ERR_NOT_EXEC	Process not being executed

Explanation

Gets IQA result with BiSCNMICRFunctionContinuously. After the reading status MF_DATARECEIVE_DONE notification, the reading results are saved in the various parameters of the MF_IQA_RESULT structure by specifying the relevant transaction number (ID) in dwTransactionNumber.

BiGetVersion

Acquires a driver version or module version.

Syntax

int **BiGetVersion***(int nDriverType, int nType, LPVERSION_INFO lpVersion)*

Argument

nDriverType: The driver regarded as the acquisition object is designated. This is an INT type.

Constant	Description
DRIVER_TYPE_J9000	TM-J9000/J9100 Driver
DRIVER_TYPE_S1000	TM-S1000 Driver

nType: The type of the version to be acquired. One of the following values can be specified. This is an INT type.

Constant	Description
VERSION_TYPE_DRIVER	Driver version
VERSION_TYPE_USB	TMUSBDriver version
VERSION_TYPE_MICR	Magnetic waveform analysis module version
VERSION_TYPE_MICR_E13B	Magnetic waveform analysis module version(E13B)
VERSION_TYPE_MICR_CMC7	Magnetic waveform analysis module version(CMC7)
VERSION_TYPE_OCR	OCR recognition module version
VERSION_TYPE_IMAGE	Image processing module version
VERSION_TYPE_IQA	IQA module version
VERSION_TYPE_BARCODE	BARCODE module version

lpVersion: Sets the address of the structure for storing the version acquisition result. This is an LPVERSION_INFO type.

VERSION_INFO structure

```
typedef struct{
CHAR lpszDescription[VERSION_CHAR_MAX];
CHAR lpszVersion[VERSION_CHAR_MAX];
}VERSION_INFO, *LPVERSION_INFO;
```

VERSION_CHAR_MAX is 64.

lpszDescription	Sets the information of the driver name
lpszVersion	Sets the version information acquired.

Return value

Value	Constant	Description
0	SUCCESS	Success
-90	ERR_PARAM	Parameter error
-220	ERR_NOT_FOUND	No data error

Explanation

Acquires this driver version or the module version used with this driver. It is possible to execute this API before executing BiOpenMonPrinter.

BiESCNEnable

Set so that scanner extended functions can be used.
Before calling BiOpenMonPrinter, it is necessary to enable scanner extended functions by calling this argument.

Syntax

int **BiESCNEnable**(*BYTE bStoreType)*

Argument

bStoreType: Select a storing method for a cropped image stored using BiESCNStoreImage. This is a BYTE type.

Constant	Value	Description
CROP_STORE_MEMORY	0	Save in memory
CROP_STORE_FILE	1	Save in a file

Return value

Value	Constant	Description
0	SUCCESS	Success
-90	ERR_PARAM	Parameter error
-160	ERR_ENABLE	Cannot be used because BiOpenMonPrinter is called

Explanation

Secure save table area with BiOpenMonPrinter.

After calling back image data reading, process the image data acquired by the device and save it in the WORK AREA.

Arguments of the scanner extended functions (BiESCN~) can be used.

Note

If this argument is called after calling BiOpenMonPrinter, the scanner extended functions cannot be used and the way of saving a cropped image cannot be changed.

BiESCNGetAutoSize

Acquire the value of capAutoSize.

Syntax

int **BiESCNGetAutoSize***(int nHandle, LPBYTE pCapAutoSize)*

Argument

nHandle: Specifies the handle. This is an INT type.

pCapAutoSize: Select a memory address to set a capAutoSize value. This is an LPBYTE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

If "1" is selected for capAutoSize, after reading image data, AutoSize processing (cut black part of the image data off) is executed, and the processed image data is saved in the WORK AREA. The width and height of the image data are automatically set to documentWidth and documentHeight.

If "0" is selected for capAutoSize, AutoSize processing and automatic setting for the width and height of the image data are not executed.

BiESCNSetAutoSize

Select the value of capAutoSize.

Syntax

int **BiESCNSetAutoSize***(int nHandle, BYTE bCapAutoSize)*

Argument

nHandle: Specifies the handle. This is an INT type.

bCapAutoSize: Select a value for a capAutoSize. This is a BYTE type.

Constant	Value	Description
CROP_AUTOSIZE_DISABLE	0	AutoSize processing disabled.
CROP_AUTOSIZE_ENABLE	1	AutoSize processing enabled.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Acquire a value of capAutoSize (AutoSize processing flag).

If an argument other than CROP_AUTOSIZE_ENABLE or CROP_AUTOSIZE_DISABLE is selected, the error (ERR_PARAM) is returned.

The AutoSize processing flag that has been set is used in the next image data reading process (AutoSize processing is not used for the image data that have been already read and saved in the WORK AREA.)

BiESCNGetCutSize

Acquires a value of CutSize. This API is compatible with the TM-J9000.

Syntax

int **BiESCNGetCutSize** (*int nHandle, LPWORD pCutSize*)

Argument

nHandle: Specifies the handle. This is an INT type.

pCutSize: *Specify the memory address to store the CutSize value in increments of 0.1 mm. This is an* LPWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The CutSize value acquired by this API is the image's left and right margins to be cropped out.

When other than zero has been specified as the CutSize, the scanned-in image is cropped to the specified size and stored in the work area.

When zero has been specified as the CutSize, cropping operation is not performed.

BiESCNSetCutSize

Sets a value of CutSize. This API is compatible with the TM-J9000.

Syntax

int **BiESCNSetCutSize** (*int nHandle, WORD wCutSize*)

Argument

nHandle: Specifies the handle. This is an INT type.

pCutSize: *Specify the width of left and right margins of an image data to be cropped out within a range of 0 to 1500 in increments of 0.1 mm. The default after executing BiOpenMonPrinter is zero. This is an* WORD type.

pCutSize	Description
0	No cropping operation is performed
1 to 1500	The image is cropped to the specified size.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Set the value specified by wCutSize to cutSize.

The cutSize is enabled only when capAutoSize has been set to CROP_AUTOSIZE_ENABLE.

Note

The specified cutSize is applied from the next scanning onward. It is not applied to images that has already been scanned and stored in the work area.

If the specified cutSize value is larger than half the width of the paper, the value is rounded down to half the width of scanned-in image.

BiESCNGetRotate

Obtains the information whether the setting for rotating image data 90° has been made to the driver. Also, obtains the value of capRotate.

Syntax

int **BiESCNGetRotate***(int nHandle, LPBYTE pCapRotate)*

Argument

nHandle: Specifies the handle. This is an INT type.

pCapRotate: Specify the address of the memory in which a return value of capRotate is stored.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Obtains the value of capRotate.

- When CROP_ROTATE_ENABLE is set to capRotate, performs rotation of the image data (rotates the image data 90° to the right or left) after reading it, and stores the edited image data in the WORK AREA.
- When CROP_ROTATE_DISABLE is set to capRotate, does not perform any rotation process.

BiESCNSetRotate

Specifies whether rotate the obtained image data 90× to the driver. Also sets a value to capRotate.

Syntax

int **BiESCNSetRotate***(int nHandle, BYTE bCapRotate)*

Argument

nHandle: Specifies the handle. This is an INT type.

bCapRotate: Specify the value for capRotate. This is a BYTE type.

Constant	Value	Description
CROP_ROTATE_ENABLE	1	Processing Rotate enabled
CROP_ROTATE_DISABLE	0	Processing Rotate disabled

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Note

The set Rotate processing flag is not applied until the scanning of the next image data. (The Rotate process is not applied to the image data already scanned and stored in the WORD AREA.)

BiESCNGetDeSkew

Obtains a threshold value of the skew angle currently set in the driver.

Syntax

int **BiESCNGetDeSkew***(int nHandle, LPWORD lpwAngle)*

Argument

nHandle: Specifies the handle. This is an INT type.

lpwAngle: Specify the address of the memory to store the threshold value of the skew angle. This is an LPWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Note

The unit of the argument is 0.01°.

BiESCNSetDeSkew

Specifies a threshold value for the skew angle to execute DeSkew.

Syntax

int **BiESCNSetDeSkew***(int nHandle, WORD wAngle)*

Argument

nHandle: Specifies the handle. This is an INT type.

wAngle: Specify a threshold value for the skew angle. (Unit: 0.01×)The following values can also be set. This is an WORD type.

Constant	Value	Description
DESKEW_ALL	0	Executes DeSkew.
DESKEW_DISABLE	65535	Does not execute DeSkew.

When a value other than DESKEW_ALL or DESKEW_DISABLE is specified and if the value is other than 1 - 8999, a parameter error occurs.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The driver reads a check and detects the skew angle. If the value exceeds the value set for this function, DeSkew is executed.
If the detected value is smaller than the one set for this function, DeSkew is not executed.
The default value for the driver is 150 (tilt of 1.5°).

Note

Even if DESKEW_All is specified, Deskew is not activated unless the following conditions are met.

- The skew angle must be 10 degrees or less.
- The entire check image must be included within the range where the image can be scanned.

BiESCNGetDocumentSize

Acquire the values of documentWidth and documentHeight. This API is compatible with the TM-J9000.

Syntax

int **BiESCNGetDocumentSize***(int nHandle, LPWORD pDocumentWidth, LPWORD pDocumentHeight)*

Argument

nHandle: Specifies the handle. This is an INT type.

pDocumentWidth: Select a memory address to set a value of the width of the image data (unit: 0.1 mm). This is an LPDWORD type.

pDocumentHeight: Select a memory address to set a value of the height of the image data (unit: 0.1 mm). This is an LPDWORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Acquire the values of documentWidth and documentHeight (the width and height of the image data saved in the WORK AREA) by using the unit of 0.1 mm.

Note

If automatic update by reading the image data or a change with BiESCNSetDocumentSize() is not executed, the default value (width=0, height=0) is acquired.

BiESCNSetDocumentSize

Sets documentWidth and documentHeight of the image data. This API is compatible with the TM-J9000.

> **Note**
> *The documentWidth and documentHeight settings made for this API are reflected on CROP_AREA_ENTIRE_IMAGE of bCropArea, as specified with BiESCNDefineCropArea.*

Syntax

int **BiESCNSetDocumentSize** (*int nHandle, WORD wDocumentWidth, WORD wDocumentHeight*)

Argument

nHandle: Specifies the handle. This is an INT type.

pDocumentWidth: This specifies the width of the image data (100 to 3000) in units of 0.1 mm. This is a WORD type.

pDocumentHeight: This specifies the height of the image data (100 to 3000) in units of 0.1 mm. This is a WORD type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

BiESCNDefineCropArea

Deletes the CropArea registration and the registered CropAreas.

Syntax

int **BiESCNDefineCropArea** (*int nHandle, BYTE bCropAreaID, WORD wStartX, WORD wStartY, WORD wEndX, WORD wEndY*)

Argument

nHandle: Specifies the handle. This is an INT type.

bCropAreaID: This specifies the CropAreaID (1 to 255) to be registered. This is a BYTE type. A user can use an ID from "2" to "255" for registering a CropArea.

Constant	Value	Description
CROP_AREA_RESET_ALL	0	All of the registered CropArea data is deleted.
CROP_AREA_ENTIRE_IMAGE	1	Zero is set for the CropArea start X and start Y coordinates, and the values of wdocumentWidth and wdocumentHeight are set for the end X and end Y coordinates.

wStartX: This specifies the start X coordinate (0 to wdocumentWidth -1) of CropArea in units of 0.1 mm. This is a WORD type.

wStartY: This specifies the start Y coordinate (0 to wdocumentHeight -1) of CropArea in units of 0.1 mm. This is a WORD type.

wEndX: This specifies the end X coordinate (1 to wdocumentWidth -1) of CropArea in units of 0.1 mm. This is a WORD type.

Constant	Value	Description
CROP_AREA_RIGHT	65535	Sets the value of wdocumentWidth as the end X coordinate

wEndY: This specifies the end Y coordinate (1 to wdocumentHeight) of CropArea in units of 0.1 mm. This is a WORD type.

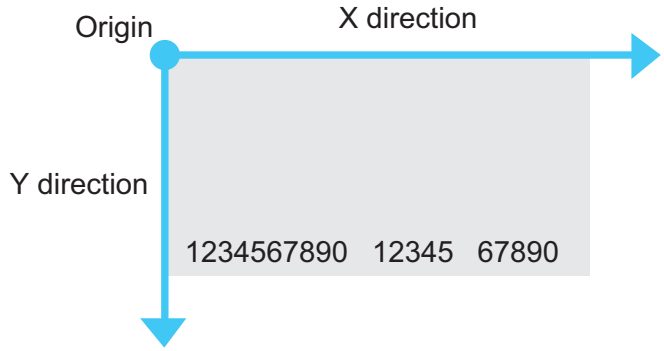
Constant	Value	Description
CROP_AREA_BOTTOM	65535	This sets the value of wdocumentHeight as the end Y coordinate

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Registers the bCropAreaID that sets the CropArea, in the CropArea definition table of the TM-S1000 API. Any specified bCropAreaID data that has already been registered is overwritten. When CROP_AREA_RESET_ALL is specified for bCropAreaID, all of the CropAreas in the CropArea definition table are deleted. The CropArea origin is the top-left corner.



Note

- ❏ Up to 255 CropAreas can be registered.
- ❏ All of the CropAreas registered in the CropArea definition table are deleted each time BiCloseMonPrinter is called. Register the CropAreas after calling BiCloseMonPrinter.
- ❏ When the start coordinate is beyond the end coordinate (Start >/= End), ERR_PARAM is returned to the return value.

BiESCNGetMaxCropAreas

Acquires the maximum supported data count that can be registered for CropArea.

Syntax

int **BiESCNGetMaxCropAreas** (*int nHandle, LPBYTE pMaxCropAreas*)

Argument

nHandle: Specifies the handle. This is an INT type.

pMaxCropAreas: This specifies the memory address in which the maximum supported data count that can be registered for CropArea is stored.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

When acquisition is successful, "255" is set in pMaxCropAreas.

BiESCNStoreImage

Crops the CropArea specified with bCropAreaID from the image data in the work area, and then saves it to either a file or memory using the save method specified with BiESCNEnable.

Syntax

int **BiESCNStoreImage** (*int nHandle, DWORD dwFileIndex, LPSTR pFileID, LPSTR pImageTagData, BYTE bCropAreaID*)

Argument

nHandle: Specifies the handle. This is an INT type.

dwFileIndex: This specifies FileIndex (an identifier) of the Crop image to be saved. NULL can also be specified. This is a DWORD type.

pFileID: This specifies FileID (an identifier) of the Crop image to be saved. This is an LPSTR type. A character string of up to 64 bytes can be specified. NULL can also be specified. Note that none of \ / : , ; * ? " < > | can be used.

pImageTagData: This specifies ImageTagData (an identifier) of the Crop image to be saved. This is an LPSTR type. A character string of up to 64 bytes can be specified. NULL can also be specified. Note that none of \ / : , ; * ? " < > | can be used.

bCropAreaID: This specifies the CropAreaID (1 to 255) registered with BiESCNDefineCropArea. This is a BYTE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-50	ERR_NO_MEMORY	Memory is insufficient
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-170	ERR_DISK_FULL	There is insufficient free space on the disk
-180	ERR_NO_IMAGE	The image data does not exist
-190	ERR_ENTRY_OVER	It is not possible to register more than the maximum allowed number of items.
-200	ERR_CROPAREAID	The specified Crop Area does not exist
-210	ERR_EXIST	The specified data has already been saved
-230	ERR_IMAGE_FILEOPEN	Open failure
-240	ERR_IMAGE_UNKNOWNFORMAT	Format injustice
-250	ERR_IMAGE_FAILED	Image data creation failed
-260	ERR_WORKAREA_NO_MEMORY	WORK AREA image creation failed due to a lack of memory
-270	ERR_WORKAREA_UNKNOWNFORMAT	WORK AREA image creation failed due to the image data format being invalid
-280	ERR_WORKAREA_FAILED	WORK AREA image creation failed
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The cropped image data has the same format as the original image data. All JPEG formats, however, are saved using standard JPEG compression.
If the size of the CropArea exceeds that of the WORK AREA image data, then the excess appears white.

The cropped image data is saved to either memory or a file, using the method specified with BiESCNEnable.
When saving to a file, the file name is formed from the device handle and each identifier (nHandle & original image data name & dwFileIndex & "_" & pFileID & "_" & pImageTagData), and is stored to the folder created by the installer.

Example:
nHandle = 1, *dwFileIndex* = 1, *pFileID* = "AA", *pImageTagData* = "BBB"

File name:

Original image data name:	File name to be stored in the Crop image save table
Image.jpg	1Image1_AA_BBB.jpg

Crop image save destination for each execution environment

- Windows2000, WindowsXP :
  Documents and Settings\All Users\EPSON\BANK\TM-S1000\Temp\
- Windows Vista or newer Windows versions :
  ProgramData\EPSON\BANK\TM-S1000\Temp\

Note

- ❏ When NULL is specified for allof the identifiers, ERR_PARAM is returned as the return value.
- ❏ The pFileID and pImageTagData arguments are not case-sensitive.Therefore, if data with the same name but in a different case already exists, then ERR_EXIST is returned as the return value.

  Example :

  When data named pFileID="AAA", pImageTagData="BBB" has already been saved, and you wish to save data named pFileID="aaa", pImageTagData="bbb", ERR_EXIST is returned as the return value.
- ❏ When saving data to a file, and a file with the same name as the save file already exists, that file is overwritten.
- ❏ All of the CropAreas registered in the CropArea definition table are deleted each time BiCloseMonPrinter is called.
- ❏ The raster format is not supported for crop image data. When raster format image data is read from a device, ERR_WORKAREA_UNKNOWNFORMAT is returned as the return value.

BiESCNRetrieveImage

Acquires the Crop image data that is saved to memory or a file.

Syntax

int **BiESCNRetrieveImage** (*int nHandle, DWORD dwFileIndex, LPSTR pFileID, LPSTR pImageTagData, LPDWORD pImageSize, LPBYTE pImageData*)

Argument

nHandle: Specifies the handle. This is an INT type.

dwFileIndex: This specifies FileIndex (an identifier) of the Crop image data to be acquired. This is a DWORD type.
While NULL can be specified, doing so prevents a search being made for the FileIndex.

pFileID: This specifies FileID (an identifier) of the Crop image data to be acquired. This is an LPSTR type. Note that none of \ / : , ; * ? " < > | can be used.
While NULL can be specified, doing so prevents a search being made for the FileID.

pImageTagData: This specifies ImageTagData (an identifier) of the Crop image data to be acquired. This is an LPSTR type. Note that none of \ / : , ; * ? " < > | can be used.
While NULL can be specified, doing so prevents a search being made for the ImageTagData.

pImageSize: Specifies the size of the memory in which the size of the acquired Crop image data is set. This is an LPDWORD type.
After calling this API, the size of the actually acquired CROP image data is returned. When there is insufficient capacity, the required size is returned together with the return value.

pImageData: Specifies the memory address where the Crop image data is set. This is an LPBYTE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-140	ERR_BUFFER_OVER_FLOW	Buffer overflow error
-220	ERR_NOT_FOUND	No data error
-230	ERR_IMAGE_FILEOPEN	Open failure
-290	ERR_IMAGE_FILEREAD	Read of the image data file failed
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

Acquires Crop image data, corresponding to the specified identifier, from the Crop image save table (memory or file) of the TM-S1000 API.
When there are multiple saved items of Crop image data corresponding to the specified identifier, only the first such item to be found is acquired.

Example :

(1) When *dwFileIndex* = 1, *pFileID* =NULL, *pImageTagData* = NULL are specified
(2) When *dwFileIndex* = 2, *pFileID* = NULL, *pImageTagData* = NULL are specified
(3) When *dwFileIndex* = NULL, *pFileID* = "B", *pImageTagData* = NULL are specified

Save order

dwFileIndex	pFileID	pImageTagData	Image Data	
1	"A"	NULL	... ...	← (1)
1	"B"	NULL	... ...	← (3)
1	"C"	NULL	... ...	
2	"A"	NULL	... ...	← (2)
2	"B"	NULL	... ...	
2	"C"	NULL	... ...	
3	"A"	"A"	... ...	
3	"B"	NULL	... ...	

Note

- ❏ All of the CropAreas registered in the CropArea definition table are deleted each time BiCloseMonPrinter is called.
- ❏ When NULL is specified for all of the identifiers, ERR_PARAM is returned as the return value.

BiESCNClearImage

Deletes the stored Crop image data.

Syntax

int **BiESCNClearImage** (*int nHandle, BYTE bFlag, DWORD dwFileIndex, LPSTR pFileID, LPSTR pImageTagData*)

Argument

nHandle: Specifies the handle. This is an INT type.

bFlag: This specifies the deletion method. This is a BYTE type. By using the deletion methods appropriately, it is possible to specify the Crop image data to be deleted. Refer to the explanation.

Macro Definition (Constant)	Value	Description
CROP_CLEAR_ALL_IMAGE	0	Deletes all the Crop image save data
CROP_CLEAR_BY_FILEINDEX	1	Deletes the Crop image data specified by dwFileIndex
CROP_CLEAR_BY_FILEID	2	Deletes the Crop image data specified by pFileID
CROP_CLEAR_BY_IMAGETAGDATA	4	Deletes the Crop image data specified by pImageTagData

dwFileIndex: This specifies FileIndex (an identifier) of the Crop image data to be deleted. This is a DWORD type.

pFileID: This specifies FileID (an identifier) of the Crop image data to be deleted. This is an LPSTR type. Note that none of \ / : , ; * ? " < > | can be used.

pImageTagData: This specifies ImageTagData (an identifier) of the Crop image data to be deleted. This is an LPSTR type. Note that none of \ / : , ; * ? " < > | can be used.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-220	ERR_NOT_FOUND	No data error
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

By using bFlag appropriately, it is possible to specify the Crop image data to be deleted.

Example : When the Crop image data specified with dwFileIndex and pImageTagData is to be deleted

bFlag = CROP_CLEAR_BY_FILEINDEX + CROP_CLEAR_BY_IMAGETAGDATA;

Note

All of the CropAreas registered in the CropArea definition table are deleted each time BiCloseMonPrinter is called.

BiESCNGetRemainingImages

Acquires the CropArea remaining count that can be registered.

Syntax

int **BiESCNGetRemainingImages** (*int nHandle, LPBYTE pRemainingImages*)

Argument

nHandle: Specifies the handle. This is an INT type.

pRemainingImages: This specifies the memory address where the CropArea remaining count that can be registered is set. This is an LPBYTE type.

Return value

Value	Constant	Description
0	SUCCESS	Success
-60	ERR_HANDLE	The handle value that specifies the device is incorrect
-90	ERR_PARAM	Parameter error
-100	ERR_NOT_SUPPORT	Unsupported
-310	ERR_EXEC_FUNCTION	Cannot be used because the other API is being executed
-400	ERR_RESET	Cannot be used because the device is being reset

Explanation

The maximum remaining count that can be acquired is 255.

Structures

MF_BASE01

```
typedef struct {
// Base Section
int iSize;                                    IN
int iVersion;                                 IN
int iRet;                                     OUT
DWORD dwNotifyType;                           IN
DWORD dwTimeout;                              IN
union {
LPHANDLE lphNotifyEvent;                      IN
HWND hNotifyWnd;                              IN
} uNotifyHandle;
HWND hProgressWnd;                            IN
WORD wErrorEject;                             IN
BYTE bBuzzerHz(MF_BUZZER_TYPE_MAX);           IN
BYTE bBuzzerCount(MF_BUZZER_TYPE_MAX);        IN
BYTE bUseNVMemory;                            IN
char cPortName(256);                          OUT
WORD wSuccessEject;                           IN
} MF_BASE01, *LPMF_BASE01;
```

- int iSize; (IN)
  This is the size of this structure.

- int iVersion; (IN)

- This is the structure version. Always specify MF_BASE_VERSION01. The MF_BASE_VERSION01 value differs for each driver version. The application uses the MF_BASE_VERSION01 for the supplied header file.

- int iRet; (OUT)
  The return value for this function is set. Regardless of its being synchronous or asynchronous, this is set when the function ends. ERR_PARAM is returned when the API is called by MF_EXEC / MF_xxxx_RETRANS without the MF_BASE01 structure being specified and when the API is called by MF_EXEC / MF_xxxx_RETRANS when only the MF_BASE01 structure is specified.

- ❑ DWORD dwNotifyType; (IN)
  When MF_EXEC is specified in the third parameter for this API, it indicates the operating method for the API. This API can be run synchronously or asynchronously. When the API is run synchronously, MF_BASE_MESSAGE_NO_MESSAGE is used, and when it is run asynchronously, another value is used.
  BiSCNMICRFunctionPostPrint and BiSCNMICRFunctionContinuously operate asynchronously regardless of the setting of dwNotifyType.
  It is run in the condition where dwNotifyType is specified to MF_BASE_MESSAGE_NO_MESSAGE, when MF_xxxx_RETRANS is specified in the third parameter for this API.

  - MF_BASE _MESSAGE_NO_MESSAGE
    When MF_EXEC / MF_xxxx_RETRANS is used in the third parameter and it is called, the function does not generate a new thread, and control returns after completion of the operation specified by the structure. (It is run synchronously.)

  - MF_BASE _MESSAGE_EVENT
    When MF_EXEC is used in the third parameter and it is called, the function runs a separate thread and immediately returns control. When the thread is started successfully, SUCCESS is returned. When there is a parameter error, invalid handle value or some other failure to meet prerequisite conditions for starting a thread, and error is returned according to the situation. When the value returned is SUCCESS, there is notification of completion of the function by setting the event handle specified by uNotifyHandle to signal status. Notification is carried out by the SetEvent function. The return value is set in iRet. When processing is canceled by BiSCNMICRCancelFunction, and event is generated. In this case, ERR_ABORT is set in iRet.

  - MF_BASE _MESSAGE_HWND
    When MF_EXEC is used in the third parameter and it is called, the function runs a separate thread and immediately returns control. When the thread is started successfully, SUCCESS is returned. When there is a parameter error, invalid handle value or some other failure to meet prerequisite conditions for starting a thread, an error is returned according to the situation. When the value returned is SUCCESS, there is notification of completion of the function in the window handle specified by uNotifyHandle. Notification is done by PostMessage, and regardless of whether it is normal or abnormal, WM_MF_DONE is sent when the function is completed. The return value is set in lParam. The return value is the same as for the API (SUCCESS, ERR_ACCESS, etc.). In addition, the same return value is set in iRet. When processing is canceled by BiSCNMICRCancelFunction, there is a WM_MF_DONE notification. In this case, ERR_ABORT is set lParam and in iRet.

  - MF_BASE _MESSAGE_BUTTON_CLICK
    When MF_EXEC is used in the third parameter and it is called, the function runs a separate thread and immediately returns control. When the thread is started successfully, SUCCESS is returned. When there is a parameter error, invalid handle value or some other failure to meet prerequisite conditions for starting a thread, an error is returned according to the situation. When the value returned is SUCCESS, there is notification of completion of the function in the button control window handle specified by uNotifyHandle using a PostMessage with WM_COMMAND (BN_CLICKED). There is WM_COMMAND (BN_CLICDED) notification when the function is completed regardless of whether it is normal or abnormal. The return value is obtained by referencing iRet. When processing is canceled by BiSCNMICRCancelFunction, there is notification with the same message. In this case, ERR_ABORT is set in iRet.

- DWORD dwTimeout (IN)
  The time to wait before paper insertion is specified in seconds. The values that can be specified are 0 to 300 (five minutes), and when 0 is specified, there is no timeout. When there is a timeout, ERR_PAPERINSERT_TIMEOUT is returned. Timeouts other than paper insertion cannot be specified. However, it is possible for the application to specify a window handle valid for hProgressWnd and manage timeout times while watching the status of message notifications using WM_MF_PROGRESS. When independent timeouts are executed, running BiSCNMICRCancelFunction is recommended. (This is not done, BiSCNMICRFunctionContinuously will continue to run, and during that time a port will be occupied.) If BiSCNMICRCancelFunction is run, the BiSCNMICRFunctionContinuously process will be interrupted, and a port will be opened.

- union uNotifyHandle; (IN)
  This sets the pointer for the event handle or the window handle for notification of completion. Since API automatically generates and discards the event handle automatically, this is not done from the application.

- HWND hProgressWnd; (IN)
  This sets a window handle for notification of progress in processing. When each data block is read during scanning, there is notification of what percentage of the total amount has been read. In the message, MF_PHASE_INIT, MF_PHASE_SCAN, MF_PHASE_MICR, MF_PHASE_PRINT or MF_PHASE_EXIT is set in wParam by WM_MF_PROGRESS. The percentage value (0-100) is set in lParam. However, even if lParam is 100%, it does not mean that this function is complete. Completion of the function is always performed by WM_MF_DONE notification. There is no problem if the window handle set by hProgressWnd and uNotifyHandle are the same. When the progress notifications unnecessary, this parameter is set 0. See "Progress Status Message List" for each of the phases, the messages sent in each phase, and the message content acquisition macros.

- WORD wErrorEject; (IN)
  This sets when and how the paper is handled when there is an error running the BiSCNMICRFunctionContinuously. There are four valid options for this member:
  MF_EXIT_ERROR_DISCHARGE specifies that the paper should be ejected to the pocket as soon as an error occurs.
  MF_EXIT_ERROR_RELEASE specifies that the paper should be unclamped and left in the same location as soon as an error occurs.
  MF_EXIT_ERROR_CONTINUE_DISCHARGE specifies that the paper should be ejected to the pocket once all other handling is complete.
  MF_EXIT_ERROR_CONTINUE_RELEASE specifies that the paper should be unclamped and left in the same location once all other handling is complete.
  If a value other then one of these is supplied ERR_PARAM is returned
  This value is ignored with BiSCNMICRFunctionContinuously.
  When MF_PROCESS structure is used, the priority is given to the action defined with MF_PROCESS.

- BYTE bBuzzerHz[MF_BUZZER_TYPE_MAX]; (IN)
  This sets the frequency of the buzzer for MICR reading. This member is made up of three array elements, and MF_BUZZER_TYPE_SUCCESS is for the case where reading is successful, MF_BUZZER_TYPE_ERROR for the case where read error is generated, and MF_BUZZER_TYPE_WFEED for the case where double feed is detected. One out of MF_BUZZER_HZ_4000, MF_BUZZER_HZ_440, and MF_BUZZER_HZ_880 is specified for each of the array elements. For example, if
  MF_BASE01. bBuzzerHz[MF_BUZZER_TYPE_SUCCESS] = MF_BUZZER_HZ_440;
  is set, a 440 Hz buzzer sounds when MICR reading is successful. Number of times the buzzer sounds can be set using bBuzzerCount. If a value other than the valid values is specified, ERR_PARAM is returned. Also, with Photo ID, this value is ignored.

- BYTE bUseNVMemory; (IN)
  Not used.

- char cPortName[256]; (OUT)
  This sets the port name. When ERR_HANDLE is generated, nothing is set. (zero clear)

- WORD wSuccessEject; (IN)
  Sets the paper ejection method in the case of successful execution of BiSCNMICRFunction / BiSCNMICRFunctionPostPrint.
  Ejects the paper into the pocket when MF_EXIT_SUCCESS_DISCHARGE is specified.
  When MF_EXIT_SUCCESS_RELEASE is specified, this only releases the paper. If values other than MF_EXIT_SUCCESS_RELEASE is specified, ERR_PARAM is returned.
  For BiSCNMICRFunctionContinuously, where the paper is ejected cannot be changed and the paper is ejected into the main pocket.

MF_MICR

```
typedef struct{
// MIC_OCR Section
int iSize;                                                    IN
int iVersion;                                                 IN
int iRet;                                                     OUT
BYTE bFont;                                                   IN
BYTE bMicOcrSelect;                                           IN
BOOL blParsing;                                               IN
BYTE bStatus;                                                 OUT
BYTE bDetail;                                                 OUT
CHAR szMicrStr(MF_MICR_CHAR_MAX);                             OUT
MF_OCR_RELIABLE_INFO stOcrReliableInfo(MF_MICR_CHAR_MAX);     OUT
CHAR szAccountNumber(MF_MICR_CHAR_MAX);                       OUT
CHAR szAmount(MF_MICR_CHAR_MAX);                              OUT
CHAR szBankNumber(MF_MICR_CHAR_MAX);                          OUT
CHAR szSerialNumber(MF_MICR_CHAR_MAX);                        OUT
CHAR szEPC(MF_MICR_CHAR_MAX);                                 OUT
CHAR szTransitNumber(MF_MICR_CHAR_MAX);                       OUT
long lCheckType;                                              OUT
long lCountryCode;                                            OUT
} MF_MICR, *LPMF_MICR;
```

- int iSize; (IN)
  This is the size of this structure.

- int iVersion; (IN)
  This is the structure version. Always specify MF_MICR_VERSION. Since the MF_MICR_VERSION value is different for each driver version, the application must always use the supplied header file.

- int iRet; (OUT)
  This sets the return values for running MICR OCR.

❏ BYTE bFont; (IN)
E13B sets MF_MICR_FONT_E13B and CMC7 sets MF_MICR_FONT_CMC7. If a value other than these is specified, ERR_PARAM is returned.
CMC7 does not support acquisition of the OCR recognition result and Parsing.

bFont : E13B

MICR character	Name	Alternate character
	Transit	t
	Amount	a
	On-Us	o
	Dash	-

bFont : CMC7

MICR character	Alternate character
	/
	#
	=
	>
	^

❏ BYTE bMicOcrSelect; (IN)
When MICR is used, MF_MICR_USE_MICR is specified; when OCR is used, MF_MICR_USE_OCR is specified, and when both are specified, MF_MICR_USE_MICR | MF_MICR_USE_OCR is specified. If a value other than these is specified, ERR_PARAM is returned. CMC7 cannot obtain the OCR recognition result. When bFont is set to MF_MICR_FONT_CMC7, set MF_MICR_USE_MICR.

❏ BOOL blParsing; (IN)
When TRUE is specified, parsing is carried out, and the character string is set in szAccountNumber, szAmount, szBankNumber, szSerialNumber, szEPC and szTransitNumber. The number is set in lCheckType, lCountryCode. FALSE is specified, parsing is not carried out. When bFont is set to MF_MICR_FONT_CMC7, set False.

❑ BYTE bStatus; (OUT)
This sets the MICR read status.

Bit	Function	Value	
		0	1
0	Reading font	E13B	CMC7
1,2	Reserved	Fixed to 0	
3	Detailed information	-	Added
4	Reread	-	Disabled
5	Reading results	Success	Failure
6	OCR processing error	No	Yes
7	Readout data reception error	No	Yes

When a scan is complete, the value is stored in the status when the first block of the image data is successfully read. For this reason, if an error occurs before the first block of the image data is successfully read, the value will not be set as the status when scan is complete.

❑ BYTE bDetail; (OUT)
This sets the MICR details read status.

Value	Information
40h	Success
41h	Check paper reading has not ever been executed. (The BiSCNMICRFunction function has not been invoked.)
44h	A delivery error occurred in the processing before reading.
45h	A magnetic waveform cannot be detected.
46h	Characters that cannot be analyzed were detected in the analysis processing.
47h	A double-feeding error or an insertion direction error occurred during check paper reading.
48h	An abnormality was detected in noise measurement.
49h	Check paper reading was stopped due to a feeding error.
4Bh	In reading, an error of paper length too long occurred.

❑ CHAR szMicrStr[MF_MICR_CHAR_MAX]; (OUT)
This sets the character string read.

❑ MF_OCR_RELIABLE_INFO stOcrReliableInfo[MF_MICR_CHAR_MAX];
This sets the candidates and reliability for the characters read.

```
typedef struct{
long lPosition;                          OUT
MF_OCR_RELIABILITY stFirstSelect;        OUT
MF_OCR_RELIABILITY stSecondSelect;       OUT
}MF_OCR_RELIABLE_INFO, *LPMF_OCR_RELIABLE_INFO;
```

❑ long lPosition; (OUT)
Position (0 is left edge).
MF_OCR_RELIABILITY stFirstSelect; (OUT)
First recognition candidate.

- MF_OCR_RELIABILITY stSecondSelect; (OUT)
  Second recognition candidate.

```
typedef struct{
char cRecogChar;                                    OUT
long lPercentage;                                   OUT
} MF_OCR_RELIABILITY *LPMF_OCR_RELIABILITY;
```

  char cRecogChar; (OUT)
  Recognized characters.
  long lPercentage; (OUT)
  Reliability (%).

- CHAR szAccountNumber[MF_MICR_CHAR_MAX];
  AccountNumber property value.

- CHAR szAmount[MF_MICR_CHAR_MAX];
  Amount property value.

- CHAR szBankNumber[MF_MICR_CHAR_MAX];
  BankNumber property value.

- CHAR szSerialNumber[MF_MICR_CHAR_MAX];
  SerialNumber property value.

- CHAR szEPC[MF_MICR_CHAR_MAX];
  EPC property value.

- CHAR szTransitNumber[MF_MICR_CHAR_MAX];
  TransitNumber property value.

- long lCheckType;
  CheckType property value.

- long lCountryCode;
  CountryCode property value.

MF_SCAN

```
typedef struct{
// Scan Section
int iSize;                          IN
int iVersion;                       IN
int iRet;                           OUT
WORD wImageID;                      IN
short sResolution;                  IN
BYTE bAddInfoDataSize;              IN
LPBYTE pAddInfoData;                IN
BYTE bStatus;                       OUT
BYTE bDetail;                       OUT
DWORD dwXSize;                      OUT
DWORD dwYSize;                      OUT
DWORD dwScanSize;                   OUT
LPBYTE lpbScanData;                 OUT
} MF_SCAN, *LPMF_SCAN;
```

- int iSize; (IN)
  This is the size of this structure.

- int iVersion; (IN)
  This is the structure version. Always specify MF_SCAN_VERSION. Since the MF_SCAN_VERSION value is different for each driver version, the application must always use the supplied header file.

- int iRet; (OUT)
  The results for scanning are set. (SUCCESS, ERR_ACCESS, etc.)

- WORD wImageID; (IN)
  This specifies the ID for an image that is read.

- short sResolution; (IN)
  This specifies the resolution for an image that is read. Select one from the following preset parameters. Values other than these, return ERR_PARAM.

sResolution	Description
MF_SCAN_DPI_DEFAULT	Reading is done at the default resolution for the device. (200 DPI)
MF_SCAN_DPI_100	Reading is at 100 DPI.
MF_SCAN_DPI_120	Reading is at 120 DPI.
MF_SCAN_DPI_200	Reading is at 200 DPI.
MF_SCAN_DPI_240	Reading is at 240 DPI.

  Small sized characters on check paper will become hard to read when MF_SCAN_DPI_100 or MF_SCAN_DPI_120 is selected.

❏ BYTE bAddInfoDataSize; (IN)
This specifies the size of additional character data. When this value is 0, no characters are added even in cases when bAddInfoData is not NULL. The maximum value that can be specified is 1024.

❏ LPBYTE pAddInfoData; (IN)
This specifies the memory address where characters to be added are set. Even when this value is not NULL, no characters are added in cases when bAddInfoDataSize is 0. The image format whose data can be added and the data format that can be added are as follows.

- TIFF : ASCII character string of the end of NULL (1 byte character only). NULL characters are not counted in the number of characters. The character string of the shorter of the following is added the value set by bAddInforDataSize, or the number of the characters from the beginning of the character string until when null appears.
- JPEG : Arbitrary binary data.

❏ BYTE bStatus; (OUT)
This sets the status when the scan is completed.

Bit	Function	Value	
		0	1
0,1,2	Reserved	Fixed to 0	
3	Rescanned	Front	Back
4	Reread	-	Disabled
5	Scanning results	Success	Failure
6	Scanning data overflow	No overflow	Not (Fixed)
7	Scanning data translation error	No error	Ends with an error

For the status when scan is complete, the value when readout of the first block of the image data is successful is stored. For this reason, if an error occurs before reading of the first block of the image data is successful, the value will not be set as the status when scan is complete.

❏ BYTE bDetail; (OUT)
This sets the detailed status when the scan is completed.

Value	Information
40h	Success
41h	No image reading result
42h	Cancellation of paper insertion waiting (Executing BiSCNMICRCancelFunction)
44h	Cancellation of image reading due to feed error
45h	Occurrence of double feed or insertion orientation error during image reading
46h	Detection of sending error before reading starts
48h	Detection of paper length error during reading

- DWORD dwXSize; (OUT)
  This sets the number of dots in the X direction for image reading.

- DWORD dwYSize; (OUT)
  This sets the number of dots in the Y direction for image reading.

- DWORD dwScanSize; (OUT)
  This sets the size of the image read.

- LPBYTE lpbScanData; (OUT)
  This sets the address of the image read. API automatically reserves and discards this memory. Therefore, the application must discard it in a timely manner. When discarding this memory on the application-side, specify this memory address for the WindowsAPI GlobalFree function

MF_PRINT01

```
typedef struct{
// Print Section
int iSize;                          IN
int iVersion;                       IN
int iRet;                           OUT
BOOL blDummy;                       IN
LPSTR lpString(3);                  IN
DWORD dwAttribute(3);               IN
WORD wFont(3);                      IN
WORD wFontSize(3);                  IN
BYTE bSpeed;                        IN
BOOL bDirection;                    IN
DWORD dwEndorseType;                IN
} MF_PRINT01, *LPMF_PRINT01;
```

- int iSize; (IN)
  This is the size of this structure.

- int iVersion; (IN)
  This is the structure version. Always specify MF_PRINT_VERSION01. Since the MF_PRINT_VERSION01 value is different for each driver version, the application must always use the supplied header file.

- int iRet; (OUT)
  This sets the return values for validation printing.

- BOOL blDummy; (IN)
  Not used.

- LPSTR lpString[3]; (IN)
  Specifies the address of the ASCII character string for electric endorse printing.
  A pointer for the character string of the first line is specified to lpString[0], a pointer for the character string of the second line is specified to lpString[1], and a pointer for the character string of the third line is specified to lpString[2]. If all are NULL pointers, ERR_PARAM is returned. “LF” may be added to the end of the last print line. When line-feeding and printing the first and second lines, the line-feeding code (CR+LF) needs to be added to the ends of the ASCII character strings.

❑ DWORD dwAttribute[3] (IN)
Specifies the attribute of the character string of the first line to dwAttribute[0], the attribute of the character string of the second line to dwAttribute[1] and the attribute of the character string of the third line to dwAttribute[2]. The attribute is specified with the following bits and multiple attributes can be specified.

Constant	Description
MF_PRINT_BOLD	Executes emphasized printing
MF_PRINT_UNDERLINE_1	Adds a 1-line width of UnderLine.
MF_PRINT_UNDERLINE_2	Adds a two-line width of UnderLine.
MF_PRINT_REVERSEVIDEO	Executes reverse printing
MF_PRINT_BLACK	Prints a character with the first color (usually it is black). (MF_PRINT_1ST_COLOR is the same.)
MF_PRINT_COLOR	Prints a character with the second color. (MF_PRINT_2ND_COLOR is the same.)
MF_PRINT_MIXED	Prints a character with the first and second colors

Bits other than the above are ignored.
When not specifying the attribute, specifies MF_PRINT_NO_ATTRIBUTE.
The line width of the underlines developed with MF_PRINT_UNDERLINE_1 and MF_PRINT_UNDERLINE_2 is the same.

❑ WORD wFont[3]; (IN)
wFont[0] specifies FONT of the character string of the first line, wFont[1] specifies FONT of the character string of the second line, and wFont[2] specifies FONT of the character string of the third line. One of the following is set for the font.

Constant	Description
MF_PRINT_FONT_A	Prints with FONT A
MF_PRINT_FONT_B	Prints with FONT B

If a value other than the above is specified, ERR_PARAM is returned.
* The values above are used only for parameter checking. The fonts used for the electric endorse depend on the operating environments.

❑ WORD wFontSize[3]; (IN)
wFontSize[0] specifies the FONT size of the character string of the first line, wFontSize[1] specifies the FONT size of the character string of the second line, and wFontSize[2] specifies the FONT size of the character string of the third line. One of the following is set.

Constant	Description
MF_PRINT_FONT_W1_H1	a font with 1 unit horizontal and 1 vertical
MF_PRINT_FONT_W1_H2	a font with 1 unit horizontal and 2 vertical
MF_PRINT_FONT_W2_H1	a font with 2 units horizontal and 1 vertical
MF_PRINT_FONT_W2_H2	a font with 2 units horizontal and 2 vertical

If a value other than the above is specified, ERR_PARAM is returned.

- ❏ BYTE bSpeed; (IN)
  Not used.

- ❏ BOOL bDirection; (IN)
  Not used.

- ❏ DWORD dwEndorseType; (IN)
  This specifies the transaction printing process and the ElectricEndorse process. One of the following values can be specified.

Constant	Description
MF_PRINT_TYPE_ELECTRIC_ENDORSE_ONLY	Executes electric endorse printing using the data specified with lpString.
MF_PRINT_TYPE_ELECTRIC_ENDORSE_EXTEND	When the electric endorsement (MF_ST_E_ENDORSEMENT, MF_ST_E_ENDORSEMENT_BACK, MF_ST_E_ENDORSEMENT_FRONT) is specified with BiSetPrintStation, after registering the size of characters or images developed with BiSetPrintSize and specifying the position of characters or images developed with BiSetPrintPosition, the electric endorse printing can be executed by executing BiPrintText, BiPrintImage, BiPrintMemoryImage.

MF_PROCESS

This structure is an option; therefore, this does not always need to be set.
When the MF_PROCESS structure is not set and reading is started, operates based on the default value of the MF_PROCESS structure; however, if the settings that correspond to the MF_BASE structure exist, the settings of the MF_BASE structure have priority.

```
typedef struct {
int iSIze;                              IN
int iVersion;                           IN
BYTE bActivationMode;                   IN
BYTE bPaperType;                        IN
DWORD dwStartWaitTime;                  IN
BYTE bSuccessStamp;                     IN
BYTE bPaperMisInsertionErrorSelect;     IN
BYTE bPaperMisInsertionErrorEject;      IN
BYTE bPaperMisInsertionStamp;           IN
BYTE bPaperMisInsertionCancel;          IN
BYTE bNoiseErrorSelect;                 IN
BYTE bNoiseErrorEject;                  IN
BYTE bNoiseStamp;                       IN
BYTE bNoiseCancel;                      IN
BYTE bDoubleFeedErrorSelect;            IN
BYTE bDoubleFeedErrorEject;             IN
BYTE bDoubleFeedStamp;                  IN
BYTE bDoubleFeedCancel;                 IN
BYTE bBaddataErrorSelect;               IN
BYTE bBaddataCount;                     IN
BYTE bBaddataErrorEject;                IN
BYTE bBaddataStamp;                     IN
BYTE bBaddataCancel;                    IN
BYTE bNodataErrorSelect;                IN
BYTE bNodataErrorEject;                 IN
BYTE bNodataStamp;                      IN
BYTE bNodataCancel;                     IN
BYTE bNearFullSelect;                   IN
BYTE bResultPartialData;                IN
} MF_PROCESS, *LPMF_PROCESS;
```

- ❏ int iSize; (IN)
  This is the size of this structure.

- ❏ int iVersion; (IN)
  This is the structure version. Always specify MF_PROCESS_VERSION. Since the MF_PROCESS_VERSION value is different for each driver version, the application must always use the supplied header file.

❏ BYTE bActivationMode; (IN)
This sets the activation mode for scanning. The valid commands are listed below.

Constant	Description
MF_ACTIVATE_MODE_HIGH_SPEED	High-speed scan mode
MF_ACTIVATE_MODE_CONFIRMATION	Confirmation scan mode

When MF_ACTIVATE_MODE_HIGH_SPEED is specified, the driver carries out all the error verification at scanning. The driver deals with errors according to the pre-set actions. The actions at error occurrence are specified in this structure. Scan speed may slow down depending on the setting for the action at error occurrence.

When MF_ACTIVATE_MODE_CONFIRMATION is specified, the application carries out all the error verification at scanning. For details of the error verification by the application, refer to the function "BiSetBehaviorToScanResult"

BYTE bPaperType; (IN)
This sets the paper type for a scan target. The valid commands are listed below.

Constant	Description
MF_PAPER_TYPE_CHECK	Check papers only
MF_PAPER_TYPE_OTHER	Papers other than check papers included

When MF_PAPER_TYPE_CHECK is specified and a paper other than check papers is scanned, a scan error may occur.
When MF_PAPER_TYPE_OTHER is selected, ease the double feed detection level. The threshold for detecting double feed can be changed using registry or setting file equivalent to the registry.The default value is listed below.

MfPaperType	Default value
MF_PAPER_TYPE_CHECK	0.17 mm
MF_PAPER_TYPE_OTHER	0.23 mm

❏ DWORD dwStartWaitTime; (IN)
This sets the wait time that is taken before the insertion starts.
The valid setting value is 0 to 6400 (unit: ms).
When a value exceeding 6400 is set, it will be rounded to 6400.

This sets the wait time that is taken from when the device has become ready for the insertion until the insertion starts.

❑ BYTE bSuccessStamp; (IN)
This sets whether to enable a Franker when scan completes successfully. The valid commands are listed below.

Constant	Description
MF_STAMP_ENABLE	Franker enabled
MF_STAMP_DISABLE	Franker disabled

The default value is MF_STAMP_DISABLE.
In the High Speed mode, operates franker in accordance with this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, operates franker in accordance with this setting value.
When bActivationMode is set to MF_ACTIVATE_MODE_HIGH_SPEED and reading is executed, if this setting is different from the following, the reading speed will slow down.
bBaddataStamp
bNodataStamp
Stamping operation cannot be shifted with Baddata, Nodata, or Success without slowing down the reading speed.

❑ BYTE bPaperMisInsertionErrorSelect; (IN)
This sets whether to detect the insertion direction error. The valid commands are listed below.

Constant	Description
MF_ERROR_SELECT_NODETECT	No error detected
MF_ERROR_SELECT_DETECT	Error detected

When MF_ERROR_SELECT_NODETECT is specified, no error is detected even if the insertion direction is wrong.
When MF_ERROR_SELECT_DETECT is specified, the action at error occurrence is taken according to the following.
bPaperMisInsertionErrorEject
bPaperMisInsertionStamp
bPaperMisInsertionCancel
For details, refer to the explanation of each element.

❑ BYTE bPaperMisInsertionErrorEject; (IN)
This sets the ejection method for when the insertion direction error is detected. The valid commands are listed below.

Constant	Description
MF_EJECT_MAIN_POCKET	Ejected to the main pocket
MF_EJECT_SUB_POCKET	Ejected to the sub pocket
MF_EJECT_NOEJECT	Not ejected (completed with error)

This setting takes place only when MF_ERROR_SELECT_DETECT is specified in bPaperMisInsertionErrorSelect.
When MF_EJECT_NOEJECT is specified, the following values are ignored.
bPaperMisInsertionStamp
bPaperMisInsertionCancel

❏ BYTE bPaperMisInsertionStamp; (IN)
This sets the whether to enable a franker for when the insertion direction error is detected. The valid commands are listed below.

Constant	Description
MF_STAMP_ENABLE	Franker enabled
MF_STAMP_DISABLE	Franker disabled

The default value is MF_STAMP_DISABLE.
If MF_ERROR_SELECT_NODETECT is set to bPaperMisInsertionErrorSelect, the setting of bPaperMisInsertionStamp is ignored.

In the High Speed mode, operates franker in accordance with this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, operates franker in accordance with this setting value.

❏ BYTE bPaperMisInsertionCancel; (IN)
Sets whether to continue reading operation when an insertion direction incorrect error occurs.
When MF_CANCEL_DISABLE is specified, does not cancel the reading process for the next check paper.
When MF_CANCEL_ENABLE is specified, cancels the reading process for the next check paper.
The default value is MF_CANCEL_DISABLE.
When MF_ERROR_SELECT_NODETECT is set to bPaperMisInsertionErrorSelect, the setting of bPaperMisInsertionCancel is ignored.
In the High Speed mode, operates in accordance with this setting value.
In the Confirmation mode, the insertion direction incorrect error is notified with MF_ERROR_OCCURRED callback and the reading process is continued. If BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, operates in accordance with this setting value.

❏ BYTE bNoiseErrorSelect; (IN)
Specifies error detection enable/disable when an external noise error is detected.
When MF_ERROR_SELECT_NODETECT is specified, an error is not detected.
When MF_ERROR_SELECT_DETECT is specified, an error is detected.
The default value is MF_ERROR_SELECT_DETECT.
When "an error not detected" is set, the following settings are ignored.
bNoiseErrorEject
bNoiseErrorStamp
NoiseErrorCancel

❏ BYTE bNoiseErrorEject;(IN
Specifies where to eject when an external noise error occurs.
Corresponds to ErrorEject of the MF_BASE structure. Even if the MF_BASE structure is set, the value of this structure has a priority.
When MF_EJECT_MAIN_POCKET is specified, paper is ejected to the main pocket.
When MF_EJECT_SUB_POCKET is specified, paper is ejected to the sub pocket.
When MF_EJECT_NOEJECT is specified, paper is not ejected.
The default value is MF_EJECT_MAIN_POCKET.
When MF_ERROR_SELECT_NODETECT is set to bBNoiseErrorSelect, the setting of bNoiseErrorEject is ignored.
In the High Speed mode, paper is ejected into the pocket in accordance with this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, paper is ejected into the pocket in accordance with this setting value.

❑ BYTE bNoiseStamp;(IN)
Specifies a franker processing when an external noise error occurs.
When MF_STAMP_ENABLE is specified, a franker is executed.
MF_STAMP_DISABLE is specified, a franker is not executed.
The default value is MF_STAMP_DISABLE.
When MF_ERROR_SELECT_NODETECT is specified to bNoiseErrorSelect, the setting of bNoiseStamp is ignored.
In the High Speed mode, executes a franker operation in accordance with this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, executes a franker operation in accordance with this setting value.

❑ BYTE bNoiseCancel;(IN)
Sets whether to continue reading when an external noise error occurs.
Corresponds to errorSelect of BiMICRSelectDataHandling. To have the compatibility with the TM-J9000 driver, when this structure is not set and the default value of this structure is not the same as the value of errorSelect, the value of errorSelect has a priority.
When MF_CANCEL_DISABLE is specified, reading for the next check paper is not canceled.
When MF_CANCEL_ENABLE is specified, reading for the next check paper is canceled.
The default value is MF_CANCEL_ENABLE.
When MF_ERROR_SELECT_NODETECT is specified to bNoiseErrorSelect, the setting of bNoiseCancel is ignored.
In the High Speed mode, executes the operation in accordance with this setting value.
In the Confirmation mode, an external noise error is notified in the MF_ERROR_OCCURRED callback and the reading process is continued. If BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, executes the operation in accordance with this setting value.

❑ BYTE bDoubleFeedErrorSelect; (IN)
This sets whether to detect the double feed error. The valid commands are listed below.

Constant	Description
MF_ERROR_SELECT_NODETECT	No error detected
MF_ERROR_SELECT_DETECT	Error detected

When MF_ERROR_SELECT_NODETECT is specified, no error is detected even if a double feed occurs.
When MF_ERROR_SELECT_DETECT is specified, the action at error occurrence is taken according to the following.
bDoubleFeedErrorEject
bDoubleFeedStamp
bDoubleFeedCancel
For details, refer to the explanation of each element.

❑ BYTE bDoubleFeedErrorEject; (IN)
This sets the ejection method for when the double feed error is detected. The valid commands are listed below.

Constant	Description
MF_EJECT_MAIN_POCKET	Ejected to the main pocket
MF_EJECT_SUB_POCKET	Ejected to the sub pocket
MF_EJECT_NOEJECT	Not ejected (completed with error)

This setting takes place only when MF_ERROR_SELECT_DETECT is specified with bDoubleFeedErrorSelect.
When MF_EJECT_NOEJECT is specified, the following values are ignored.
bDoubleFeedStamp
bDoubleFeedCancel

❑ BYTE bDoubleFeedStamp; (IN)
This sets whether to enable franker when the double feed error is detected. The valid commands are listed below.

Constant	Description
MF_STAMP_ENABLE	Franker enabled
MF_STAMP_DISABLE	Franker disabled

The default value is MF_STAMP_DISABLE.
When MF_ERROR_SELECT_NODETECT is specified to bDoubleFeedErrorSelect, the setting of bDoubleFeedStamp is ignored.
In the High Speed mode, a franker operation is executed in accordance with this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, a franker operation is executed in accordance with this setting value.

❑ BYTE bDoubleFeedCancel; (IN)
Sets whether to continue reading when a double-feed error occurs.
Corresponds to errorSelect of BiMICRSelectDataHandling.
When this structure is not set and the default value of this structure is not the same as the value of errorSelect, the value of errorSelect has a priority.
When MF_CANCEL_DISABLE is specified, reading process for the next check paper is not canceled.
When MF_CANCEL_ENABLE is specified, reading process for the next check paper is canceled.
The default value is MF_CANCEL_DISABLE.
When MF_ERROR_SELECT_NODETECT is specified to bDoubleFeedErrorSelect, the setting of bDoubleFeedCancel is ignored.
In the High Speed mode, the operation is executed in accordance with this setting value.
In the Confirmation mode, the double-feed error is notified in the MF_ERROR_OCCURRED callback notification and the reading process is continued. If BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, executes the operation in accordance with this setting value.

❑ BYTE bBaddataErrorSelect; (IN)
This sets whether to detect characters that cannot be recognized by Mirth. The valid commands are listed below.

Constant	Description
MF_ERROR_SELECT_NODETECT	No error detected
MF_ERROR_SELECT_DETECT	Error detected

The default value is MF_ERROR_SELECT_ DETECT.
When "No error detected" is set, the following settings are ignored.
bBaddataCount
bBaddataErrorEject
bBaddataStamp
bBaddataCancel

❑ BYTE bBaddataCount; (IN)
This sets the permissible number of characters for when the MICR character recognition error is detected.
The valid setting value is 0 to 255.

When MICR character recognition error is detected and the number of unrecognized characters exceeds the permissible number of this setting, the action at error occurrence is taken according to the following.
bBaddataStamp
bBaddataErrorEject
bBaddataCancel
For details, refer to the explanation of each element.

When the value is set to 0, the action at error occurrence is not taken even if the MICR character recognition error occurs. When the value is set to 255, all the actions at error occurrence are taken if the MICR character recognition error occurs.

❑ BYTE bBaddataErrorEject; (IN)
This sets the ejection method for when the MICR character recognition error is detected and the number of characters detected exceeds the permissible number. The valid commands are listed below.

Constant	Description
MF_EJECT_MAIN_POCKET	Ejected to the main pocket
MF_EJECT_SUB_POCKET	Ejected to the sub pocket
MF_EJECT_NOEJECT	Not ejected (completed with error)

The default value is MF_EJECT_MAIN_POCKET.
When bBaddataErrorSelect is specified to MF_ERROR_SELECT_NODETECT or when the number of characters that cannot be analyzed is not overfewer than the permissible number specified with bBaddataCount, the setting of bBaddataErrorEject is ignored.
In the High Speed mode, paper is ejected to the pocket corresponding to this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, paper is ejected to the pocket corresponding to this setting value.
When MF_ACTIVATE_MODE_HIGH_SPEED is set to bActivationMode, the scan speed will slow down if a value other than MF_EJECT_MAIN_POCKET is specified in this setting.

❑ BYTE bBaddataStamp; (IN)
This sets whether to enable a franker for when the MICR character recognition error is detected and the number of error characters exceeds the permissible number. The valid commands are listed below.

Constant	Description
MF_STAMP_ENABLE	Franker enabled
MF_STAMP_DISABLE	Franker disabled

This setting takes place only when MF_ERROR_SELECT_DETECT is specified in bBaddataErrorSelect and the number of unrecognized characters exceeds the permissible number set in bBaddataCount.
When MF_ACTIVATE_MODE_HIGH_SPEED is set to bActivationMode, the scan speed will slow down if this setting is different from the following.
bSuccessStamp
bNodataStamp

❑ BYTE bBaddataCancel; (IN)
This sets whether to cancel the action for when the MICR character recognition error is detected and the number of error characters exceeds the permissible number. The valid commands are listed below.

Constant	Description
MF_CANCEL_DISABLE	Reading process for the next check paper is not canceled
MF_CANCEL_ENABLE	Reading process for the next check paper is canceled

The default value is MF_CANCEL_DISABLE.
When bBaddataErrorSelect is specified to MF_ERROR_SELECT_NODETECT or when the number of characters that cannot be analyzed is fewer than the permissible number specified with bBaddataCount, the setting of bBaddataCancel is ignored.
In the High Speed mode, executes the operation corresponding to this setting value.
In the Confirmation mode, if the number of characters that cannot be analyzed is over the permissible number, it is notified in the MF_ERROR_OCCURRED callback and the reading process is continued.
If BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, executes the operation corresponding to this setting value.
When MF_ACTIVATE_MODE_HIGH_SPEED is set to bActivationMode, the scan speed will slow down if MF_CANCEL_ENABLE is specified in this setting.

❑ BYTE bNodataErrorSelect; (IN)
This sets whether to detect errors when magnetic waveform is not found. The valid commands are listed below.

Constant	Description
MF_ERROR_SELECT_NODETECT	No error detected
MF_ERROR_SELECT_DETECT	Error detected

When MF_ERROR_SELECT_NODETECT is specified, no error is detected even if no MICR magnetic waveform is found.
When MF_ERROR_SELECT_DETECT is specified, the action at error occurrence is taken according to the following.
bNodataErrorEject
bNodataStamp
bNodataCancel
For details, refer to the explanation of each element.

❑ BYTE bNodataErrorEject; (IN)
This sets the ejection method for when an error is detected because MICR magnetic waveform is not found. The valid setting values are as shown below.

Constant	Description
MF_EJECT_MAIN_POCKET	Ejected to the main pocket
MF_EJECT_SUB_POCKET	Ejected to the sub pocket
MF_EJECT_NOEJECT	Not ejected (completed with error)

This setting takes place only when MF_ERROR_SELECT_DETECT is specified in bNodataErrorSelect.
When MF_EJECT_NOEJECT is specified, the following values are ignored.
bNodataStamp
bNodataCancel

When MF_ACTIVATE_MODE_HIGH_SPEED is set to bActivationMode, the scan speed will slow down if a value other than MF_EJECT_MAIN_POCKET is specified in this setting.

❑ BYTE bNodataStamp; (IN)
This sets whether to enable a franker for when an error is detected because MICR magnetic waveform is not found. The valid commands are listed below.

Constant	Description
MF_STAMP_ENABLE	Franker enabled
MF_STAMP_DISABLE	Franker disabled

This setting takes place only when MF_ERROR_SELECT_DETECT is specified in bNodataErrorSelect.
When MF_ACTIVATE_MODE_HIGH_SPEED is set to bActivationMode, the scan speed will slow down if this setting is different from the following.
bSuccessStamp
bBaddataStamp

❏ BYTE bNodataCancel; (IN)
This sets whether to cancel the action for when an error is detected because that MICR magnetic waveform is not found. The valid commands are listed below.

Constant	Description
MF_CANCEL_DISABLE	Reading process for the next check paper is not canceled
MF_CANCEL_ENABLE	Reading process for the next check paper is canceled

The default value is MF_CANCEL_DISABLE.
When bNodataErrorSelect is specified to MF_ERROR_SELECT_NODETECT, the setting of bNodataCancel is ignored.
In the High Speed mode, executes the operation corresponding to this setting value.
In the Confirmation mode, an MICR magnetic waveform undetected error is notified in the MF_ERROR_OCCURRED callback, and the reading process is continued. If BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, executes the operation corresponding to this setting value.
When MF_ACTIVATE_MODE_HIGH_SPEED is set to bActivationMode, the scan speed will slow down if MF_CANCEL_ENABLE is specified in this setting.

❏ BYTE bNearFullSelect; (IN)
This sets whether to permit scanning when the eject pocket is nearly full. The valid commands are listed below.

Constant	Description
MF_NEARFULL_PERMIT	Scan permitted
MF_NEARFULL_MAIN_PERMIT	Scan permitted(Main pocket)
MF_NEARFULL_SUB_PERMIT	Scan permitted(Sub pocket)
MF_NEARFULL_NOT_PERMIT	Scan not permitted

When MF_NEARFULL_NOT_PERMIT is specified, scanning stops if the ejection pocket is found nearly full.

Note

Do not use MF_NEARFULL_MAIN_PERMIT and MF_NEARFULL_SUB_PERMIT when the Waterfall mode is executed.

❏ BYTE bResultPartialData; (IN)
This sets whether to acquire data that is being scanned when an error that does not interrupt any operation occurs in the middle of the scanning. The valid commands are listed below.

Constant	Description
MF_RESULT_PARTIAL	Data being scanned can be acquired
MF_RESULT_NONE	Data being scanned is deleted

MF_OCR_AB

```
typedef struct {
int iSize;                                                          IN
int iVersion;                                                       IN
int iRet;                                                           OUT
BYTE bOcrType;                                                      IN
BYTE bDirection;                                                    IN
WORD wStartX;                                                       IN
WORD wStartY;                                                       IN
WORD wEndX;                                                         IN
WORD wEndY;                                                         IN
BYTE bSpeceHandling;                                                IN
CHAR szOcrStr(MF_OCR_AB_CHAR_MAX);                                  OUT
MF_OCR_RELIABLE_INFO strOcrReliableInfo (MF_OCR_AB_CHAR_MAX);       OUT
} MF_OCR_AB, *LPMF_OCR_AB;
```

- ❏ int iSize;(IN)
  Specifies the size of this structure.

- ❏ int iVersion;(IN)
  Specifies the version of this structure. Be sure to specify MF_OCR_AB_VERSION.

- ❏ int iRet;(OUT)
  Stores the return value of the OCR recognition processing.

- ❏ BYTE bOcrType;(IN)
  Specifies the font type. One of the following can be specified.

Constant	Description
MF_OCR_FONT_OCRA_NUM	OCR-A font and numbers only
MF_OCR_FONT_OCRB_NUM	OCR-B font and numbers only
MF_OCR_FONT_OCRA_ALPHA	OCR-A font and alphabetic characters only
MF_OCR_FONT_OCRB_ALPHA	OCR-B font and alphabetic characters only
MF_OCR_FONT_OCRA_ALPHANUM	OCR-A font and alphanumeric characters
MF_OCR_FONT_OCRB_ALPHANUM	OCR-B font and alphanumeric characters
MF_OCR_FONT_OCRA_ALPHANUM_WOOH	OCR-A font and alphanumeric characters (except for OH.)
MF_OCR_FONT_OCRB_ALPHANUM_WOOH	OCR-B font and alphanumeric characters (except for OH.)
MF_OCR_FONT_OCRA_ALPHANUM_WOZERO	OCR-A font and alphanumeric characters (except for ZERO.)
MF_OCR_FONT_OCRB_ALPHANUM_WOZERO	OCR-B font and alphanumeric characters (except for ZERO.)
MF_OCR_FONT_OCRA_SYMNUM	OCR-A font, numbers, and symbols (excluding "+")
MF_OCR_FONT_OCRB_SYMNUM	OCR-B font, numbers, and symbols (including "+")

- BYTE bDirection;(IN)
  Specifies the character direction for the area for which the OCR recognition is executed. One of the following values can be specified.

Constant	Description
MF_OCR_LEFTRIGHT	From left to right (normal direction)
MF_OCR_TOPBOTTOM	From top to bottom (90-clockwise rotation)
MF_OCR_RIGHTLEFT	From right to left (flip vertical)
MF_OCR_BOTTOMTOP	From bottom to top (90-counterclockwise rotation)

- WORD wStartX;(IN)
  Specifies the starting point X-coordinate of the area for which the OCR recognition is executed (Units: mm). The available range is 0 to 254.

- WORD wStartY;(IN)
  Specifies the starting point Y-coordinate of the area for which the OCR recognition is executed (Units: mm). The available range is 0 to 254.

- WORD wEndX;(IN)
  Specifies the ending point X-coordinate of the area for which the OCR recognition is executed (Units: mm). The available range is 1 to 255.
  When OCR_AREA_RIGHT is specified, the right side of an image can be specified.
  When OCR_AREA_LEFT is specified, the left side of an image can be specified.

- WORD wEndY;(IN)
  Specifies the ending point Y-coordinate of the area for which the OCR recognition is executed (Units: mm). The available range is 1 to 255.
  When OCR_AREA_BOTTOM is specified, the bottom side of an image can be specified.
  When OCR_AREA_TOP is specified, the top side of an image can be specified.
  When the ending point is specified to the origin, all the area for which the OCR recognition is executed can be specified. In this case, the character string is analyzed as one line.

- BYTE bSpaceHandling;(IN)
  Specifies a handling method of space characters for the OCR recognition processing.
  When OCR_SPACE_ENABLE is specified, space characters are included in the OCR recognition result.
  When OCR_SPACE_DISABLE is specified, space characters are not included in the OCR recognition result.

- CHAR szOcrStr[MF_OCR_AB_CHAR_MAX];(OUT)
  Stores character strings acquired by the OCR recognition processing.
  This character string is generated from the first candidate of each character.
  The value of MF_OCR_AB_CHAR_MAX is 128.

- MF_OCR_RELIABLE_INFO stOcrReliableInfo[MF_OCR_AB_CHAR_MAX];(OUT)
  Sets the candidatecandidates and the reliability reliabilities for the first and the second characters acquired by the OCR recognition processing. Character types and reliabilities for the first and the second candidates can be acquired at the same time as shown in the example below:

  ```
  stFirstSelect.cRecogChar;      0
  stFirstSelect.lPercentage;     80%
  stSecondSelect.cRecogChar;     0
  stSecondSelect.lPercentage;    20%
  ```

  For MF_OCR_RELIABLE_INFO, refer to "MF_OCR_RELIABLE_INFO" on page 4-164.

MF_OCR_RELIABILITY

```
typedef struct {
char cRecogChar;                                    OUT
long lPercentage;                                   OUT
} MF_OCR_RELIABILITY *LPMF_OCR_RELIABILITY;
```

- ❏ char cRecogChar; (OUT)
  Recognized characters

- ❏ long lPercentage; (OUT)
  Reliability (%)

MF_OCR_RELIABLE_INFO

```
typedef struct {
long lPosition;                                     OUT
MF_OCR_RELIABILITY stFirstSelect;                   OUT
MF_OCR_RELIABILITY stSecondSelect;                  OUT

} MF_OCR_RELIABLE_INFO, *LPMF_OCR_RELIABLE_INFO;
```

- ❏ long lPosition; (OUT)
  Position (0 is far left.)

- ❏ MF_OCR_RELIABILITY stFirstSelect; (OUT)
  The first choice for the recognition.

- ❏ MF_OCR_RELIABILITY stSecondSelect; (OUT)
  The second choice for the recognition.

MF_IQA

```
typedef struct {
int iSize;                                  IN
int iVersion;                               IN
BYTE bErrorSelect;                          IN
BYTE bErrorEject;                           IN
BYTE bStamp;                                IN
BYTE bCancel;                               IN
BYTE bImageFormat;                          IN
BYTE bColorDepth;                           IN
CHAR bThreshold;                            IN
BYTE bColor;                                IN
BYTE bExOption;                             IN
short sResolution;                          IN
BYTE bUndersize;                            IN
BYTE bOversize;                             IN
BYTE bMincompressed;                        IN
BYTE bMaxcompressed;                        IN
BYTE bFront_rear;                           IN
BYTE bToolight;                             IN
BYTE bToodark;                              IN
BYTE bStreaks;                              IN
BYTE bNoise;                                IN
BYTE bFocus;                                IN
BYTE bCorners;                              IN
BYTE bEdges;                                IN
BYTE bFraming;                              IN
BYTE bSkew;                                 IN
BYTE bCarbon;                               IN
BYTE bPiggyback;                            IN

} MF_IQA, *LPMF_IQA;
```

- int iSize; (IN)
  This is the size of this structure.

- int iVersion; (IN)
  This is the structure version. Always specify MF_IQA_VERSION.
  Since the MF_IQA_VERSION value is different for each driver version, the application must always use the supplied header file.

❑ BYTE bErrorSelect;(IN)
This sets whether to detect the image quality defects (IQA error). The valid commands are listed below.

Constant	Description
MF_ERROR_SELECT_NODETECT	No error detected
MF_ERROR_SELECT_DETECT	Error detected

When MF_ERROR_SELECT_NODETECT is specified, no error is detected even if the insertion direction is wrong.
When MF_ERROR_SELECT_DETECT is specified, the action at error occurrence is taken according to the following.
bErrorEject
bStamp
bCancel
For details, refer to the explanation of each element.

❑ BYTE bErrorEject;(IN)
This sets the ejection method for when IQA error is detected. The valid commands are listed below.

Constant	Description
MF_EJECT_MAIN_POCKET	Ejected to the main pocket
MF_EJECT_SUB_POCKET	Ejected to the sub pocket
MF_EJECT_NOEJECT	Not ejected (completed with error)

This setting takes place only when MF_ERROR_SELECT_DETECT is specified in bErrorSelect.
This setting has a priority even when wErrorEject of MF_BASE structure is set.
In the High Speed mode, paper is ejected to the pocket set with this setting.
In the Confirmation mode, when BiSetBehaviorToScnResult is not called in the MF_DATARECEIVE_DONE callback notification, paper is ejected to the pocket set with this setting.
When reading is processed in the High Speed mode, if this setting is set to other than MF_EJECT_MAIN_POCKET, reading speed slows down.

❑ BYTE bStamp;(IN)
This sets the whether to enable a franker for when the IQA error is detected. The valid commands are listed below.

Constant	Description
MF_STAMP_ENABLE	Franker eabled
MF_STAMP_DISABLE	Franker disabled

The default value is MF_STAMP_DISABLE.
If MF_ERROR_SELECT_NODETECT is set to bErrorSelect, the setting of bStamp is ignored.
In the High Speed mode, operates franker in accordance with this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, operates franker in accordance with this setting value.
If the setting for bSuccessStamp of MF_PROCESS structure differs from this setting when reading is processed in the High Speed mode, reading speed slows down.

❏ BYTE bCancel; (IN)
Sets whether to continue reading operation when the IQA error occurs.

Constant	Description
MF_CANCEL_DISABLE	Does not cancel the reading process for the next check paper.
MF_CANCEL_ENABLE	Cancels the reading process for the next check paper.

The default value is MF_CANCEL_DISABLE.
When MF_ERROR_SELECT_NODETECT is set to bErrorSelect, the setting of bCancel is ignored.
In the High Speed mode, operates in accordance with this setting value.
In the Confirmation mode, the insertion direction incorrect error is notified with MF_ERROR_OCCURRED callback and the reading process is continued. If BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, operates in accordance with this setting value.
When reading is processed in the High Speed mode, if this setting is set to other than MF_CANCEL_ENABLE, reading speed slows down.

❏ BYTE bImageFormat;(IN)
This sets an image format at the IQA validation. For details of the setting value, see "BiSCNSetImageFormat" on page 4-34. The default value is EPS_BI_SCN_TIFF.

❏ BYTE bColorDepth;(IN)
This sets the gradation (bits per pixel) at the IQA validation. For details of the setting value, see bColorDepth in "BiSCNSetImageQuality" on page 4-32. The default value is EPS_BI_SCN_1BIT.

❏ CHAR bThreshold;(IN)
This sets the density threshold at the IQA validation. Enabled when bColorDepth is set to EPS_BI_SCN_1BIT, and bExOption is set to EPS_BI_SCN_MANUAL.
The valid value is -128 to 127. The default value is "0."

❏ BYTE bColor;(IN)
This sets color at the IQA validation. For details of the setting value, see bColor in "BiSCNSetImageQuality" on page 4-32. The default value is EPS_BI_SCN_MONOCHROME.

❏ BYTE bExOption;(IN)
This sets the variety of density adjustment at the IQA validation. For details of the setting value, see bExOption in "BiSCNSetImageQuality" on page 4-32.

❏ short sResolution; (IN)
This sets the resolution at the IQA validation. For details of the setting value, see sResolution in "MF_SCAN" on page 4-146.

❏ BYTE bUndersize; (IN)
This sets the execution of UndersizeImage validation. The valid commands are listed below. The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❑ BYTE bOversize; (IN)
This sets the execution of OversizeImage validation. The valid commands are listed below. The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❑ BYTE bMincompressed; (IN)
This sets the execution of MinCompressedImageSize validation. The valid commands are listed below. The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

Note

When the combination of bColorDepth and bImageFormat is other than the ones listed below, MinCompressedImageSize validation is not executed even if this is set to MF_IQA_TEST_ENABLE.

bColorDepth	bImageFormat
EPS_BI_SCN_1BIT	EPS_BI_SCN_TIFF
	EPS_BI_SCN_JPEGHIGH
	EPS_BI_SCN_JPEGNORMAL
	EPS_BI_SCN_JPEGLOW
	EPS_BI_SCN_JTIFF
EPS_BI_SCN_8BIT	EPS_BI_SCN_JPEGHIGH
	EPS_BI_SCN_JPEGNORMAL
	EPS_BI_SCN_JPEGLOW
	EPS_BI_SCN_JTIFF

❑ BYTE bMaxcompressed; (IN)
This sets the execution of MaxCompressedImageSize validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

Note

When the image format is not a compression format, MaxCompressedImageSize validation is not executed even if this is set to MF_IQA_TEST_ENABLE. For details, refer to *bMincompressed (4-168)*.

❏ BYTE bFront_rear; (IN)
This sets the execution of FrontRearImageMismatch validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❏ BYTE bToolight; (IN)
This sets the execution of ImageTooLight validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❏ BYTE bToodark; (IN)
This sets the execution of ImageTooDark validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❏ BYTE bStreaks; (IN)
This sets the execution of HorizontalStreaksPresent validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❏ BYTE bNoise; (IN)
This sets the execution of ExcessiveSpotNoise validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

Note

When bColorDepth is set to EPS_BI_SCN_8BIT, ExcessiveSpotNoise validation is not executed even if this is set to MF_IQA_TEST_ENABLE.

❑ BYTE bFocus; (IN)
This sets the execution of ImageOutOfFocus validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

Note

When bColorDepth is set to EPS_BI_SCN_1BIT, ImageOutOfFocus validation is not executed even if this is set to MF_IQA_TEST_ENABLE.

❑ BYTE bCorners; (IN)
This sets the execution of FoldedTornDocCorners validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❑ BYTE bEdges; (IN)
This sets the execution of FoldedTornDocEdges validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❑ BYTE bFraming; (IN)
This sets the execution of DocFramingError validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❑ BYTE bSkew; (IN)
This sets the execution of ExcessiveDocSkew validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❏ BYTE bCarbon; (IN)
This sets the execution of CarbonStripDetection validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

❏ BYTE bPiggyback; (IN)
This sets the execution of Piggyback validation. The valid commands are listed below.
The default value is MF_IQA_TEST_DISABLE.

Constant	Description
MF_IQA_TEST_DISABLE	Validation is not executed.
MF_IQA_TEST_ENABLE	Validation is executed.

MF_IQA_RESULT

```
typedef struct {
int iSize;                                                          IN
int iVersion;                                                       IN
int iRet;                                                           OUT
IQARESULT_UNDERSIZE_IMAGE stUnderSize;                              OUT
IQARESULT_OVERSIZE_IMAGE stOverSize;                                OUT
IQARESULT_MIN_COMPRESSED_IMAGE_SIZE stMinCompressedImageSize;       OUT
IQARESULT_MAX_COMPRESSED_IMAGE_SIZE stMaxCompressedImageSize;       OUT
IQARESULT_FRONT_REAR_IMAGE_MISMATCH stFrontRearImageMismatch;       OUT
IQARESULT_IMAGE_TOO_LIGHT stImageTooLight;                          OUT
IQARESULT_IMAGE_TOO_DARK stImageTooDark;                            OUT
IQARESULT_HORIZONTAL_STREAKS_PRESENT stHorizontalStreaksPresent;    OUT
IQARESULT_EXCESSIVE_SPOT_NOISE stExcessiveSpotNoise;                OUT
IQARESULT_IMAGE_OUT_OF_FOCUS stImageOutOfFocus;                     OUT
IQARESULT_FOLDED_TORN_DOC_CORNERS stFoldedTornDocCorners;           OUT
IQARESULT_FOLDED_TORN_DOC_EDGES stFoldedTornDocEdges;               OUT
IQARESULT_DOC_FRAMING_ERROR stDocFramingError;                      OUT
IQARESULT_EXCESSIVE_DOC_SKEW stExcessiveDocSkew;                    OUT
IQARESULT_CARBON_STRIP_DETECTION stCarbonStripDetection;            OUT
IQARESULT_PIGGYBACK stPiggyBack;                                    OUT

} MF__IQA_RESULT, *LPMF_IQA_RESULT;
```

- int iSize; (IN)
  This is the size of this structure.

- int iVersion; (IN)
  This is the structure version. Always specify MF_IQARESULT_VERSION.
  Since the MF_IQARESULT_VERSION value is different for each driver version, the application must always use the supplied header file.

- int iRet; (OUT)
  This sets the return values for running IQA.

- stUnderSize; (OUT)
  UndersizeImage validation result is set.

```
typedef struct tag_IQA_UNDERSIZE_IMAGE {
    BYTE bResult;       // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iWidth;         // width in tenths of an inch
    int iHeight;        // height in tenths of an inch
} IQARESULT_UNDERSIZE_IMAGE, *LPIQARESULT_UNDERSIZE_IMAGE;
```

- stOverSize; (OUT)
  OversizeImage validation result is set.

```
typedef struct tag_IQA_OVERSIZE_IMAGE {
    BYTE bResult;       // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iWidth;         // width in tenths of an inch
    int iHeight;        // height in tenths of an inch
} IQARESULT_OVERSIZE_IMAGE, *LPIQARESULT_OVERSIZE_IMAGE;
```

❏ stMinCompressedImageSize; (OUT)
MinCompressedImageSize validation result is set.

```
typedef struct tag_IQA_MIN_COMPRESSED_IMAGE_SIZE {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iSize;             // compressed image size in bytes
} IQARESULT_MIN_COMPRESSED_IMAGE_SIZE,
 *LPIQARESULT_MIN_COMPRESSED_IMAGE_SIZE;
```

❏ stMaxCompressedImageSize; (OUT)
MaxCompressedImageSize validation result is set.

```
typedef struct tag_IQA_MAX_COMPRESSED_IMAGE_SIZE {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iSize;             // compressed image size in bytes
} IQARESULT_MIN_COMPRESSED_IMAGE_SIZE,
 *LPIQARESULT_MIN_COMPRESSED_IMAGE_SIZE;
```

❏ stFrontRearImageMismatch; (OUT)
FrontRearImageMismatch validation result is set.

```
typedef struct tag_IQA_FRONT_REAR_IMAGE_MISMATCH {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iAbsWidthDiff;     // absolute value of width difference between front and rear
                              in tenths of an inch
    int iAbsHeightDiff;    // absolute value of height difference between front and rear
                              in tenths of an inch
} IQARESULT_FRONT_REAR_IMAGE_MISMATCH,
 *LPIQARESULT_FRONT_REAR_IMAGE_MISMATCH;
```

❏ stImageTooLight; (OUT)
ImageTooLight validation result is set.

```
typedef struct tag_IQA_IMAGE_TOO_LIGHT {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iBlackPixels;      // percenage of black pixels in the image in units of 0.1 percent
    int iBrightness;       // percent image brightness in units of 0.1 percent
    int iContrast;         // percent image contrast in units of 0.1 percent
} IQARESULT_IMAGE_TOO_LIGHT,*LPIQARESULT_IMAGE_TOO_LIGHT;
```

❏ stImageTooDark; (OUT)
ImageTooDark validation result is set.

```
typedef struct tag_IQA_IMAGE_TOO_DARK {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iBlackPixels;      // percenage of black pixels in the image in units of 0.1 percent
    int iBrightness;       // percent image brightness in units of 0.1 percent
} IQARESULT_IMAGE_TOO_DARK,*LPIQARESULT_IMAGE_TOO_DARK;
```

❑ stHorizontalStreaksPresent; (OUT)
HorizontalStreaksPresent validation result is set.

```
typedef struct tag_IQA_HORIZONTAL_STREAKS_PRESENT {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iStreakCount;      // Number of black streaks present in bitonalimages
    int iStreakHeight;     // Height of the largest horizontal black streak
} IQARESULT_HORIZONTAL_STREAKS_PRESENT,
 *LPIQARESULT_HORIZONTAL_STREAKS_PRESENT;
```

❑ stExcessiveSpotNoise; (OUT)
ExcessiveSpotNoise validation result is set.

```
typedef struct tag_IQA_EXCESSIVE_SPOT_NOISE {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iCount;            // average number of spots per square inch of the  image
                              defined for bi-tonal only
} IQARESULT_EXCESSIVE_SPOT_NOISE, *LPIQARESULT_EXCESSIVE_SPOT_NOISE;
```

❑ stImageOutOfFocus; (OUT)
ImageOutOfFocus validation result is set.

```
typedef struct tag_IQA_IMAGE_OUT_OF_FOCUS {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iImageFocusScore;
                    // (max video gradient) /(grey level dynamic range) * (pixel pitch)
} IQARESULT_IMAGE_OUT_OF_FOCUS, *LPIQARESULT_IMAGE_OUT_OF_FOCUS;
```

❑ stFoldedTornDocCorners; (OUT)
FoldedTornDocCorners validation result is set.

```
typedef struct tag_IQA_FOLDED_TORN_DOC_CORNERS {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iTopLeftWidth;     // Dimensions of circumscribing rectangle of torn/folded
    int iTopLeftHeight;       corners in tenths of inches
    int iTopRightWidth;
    int iTopRightHeight;
    int iBottomLeftWidth;
    int iBottomLeftHeight;
    int iBottomRightWidth;
    int iBottomRightHeight;
} IQARESULT_FOLDED_TORN_DOC_CORNERS,
 *LPIQARESULT_FOLDED_TORN_DOC_CORNERS;
```

❏ stFoldedTornDocEdges; (OUT)
FoldedTornDocEdges validation result is set.

```
typedef struct tag_IQA_FOLDED_TORN_DOC_EDGES {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iTopWidth;         // Dimensions of circumscribing rectangle of  torn/folded
    int iTopHeight;           corners in tenths of inches
    int iLeftWidth;
    int iLeftHeight;
    int iRightWidth;
    int iRightHeight;
    int iBottomWidth;
    int iBottomHeight;
} IQARESULT_FOLDED_TORN_DOC_EDGES,
 *LPIQARESULT_FOLDED_TORN_DOC_EDGES;
```

❏ stDocFramingError; (OUT)
DocFramingError validation result is set.

```
typedef struct tag_IQA_DOC_FRAMING_ERROR {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iTop;              // Dimensions of the additional scanlines on each side of
    int iLeft;                the frame in tenths of an inch
    int iRight;
    int iBottom;
} IQARESULT_DOC_FRAMING_ERROR, *LPIQARESULT_DOC_FRAMING_ERROR;
```

❏ stExcessiveDocSkew; (OUT)
ExcessiveDocSkew validation result is set.

```
typedef struct tag_PIQA_EXCESSIVE_DOC_SKEW {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iAngle;            // Angle of skew in tenths of a degree
    int iRange;            // Fixed to 0
} IQARESULT_EXCESSIVE_DOC_SKEW, *LPIQARESULT_EXCESSIVE_DOC_SKEW;
```

❏ stCarbonStripDetection; (OUT)
CarbonStripDetection validation result is set.

```
typedef struct tag_IQA_CARBON_STRIP_DETECTION {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
    int iStripHeight;      // Carbon strip height
} IQARESULT_CARBON_STRIP_DETECTION,
 *LPIQARESULT_CARBON_STRIP_DETECTION;
```

❏ stPiggyBack; (OUT)
Piggyback validation result is set.

```
typedef struct tag_IQA_PIGGYBACK {
    BYTE bResult;          // test result (Refer to "BiGetIQAResult" on page 4-113)
} IQARESULT_PIGGYBACK, *LPIQARESULT_PIGGYBACK;
```

MF_BARCODE

```
typedef struct {
int iSize;                                          IN
int iVersion;                                       IN
int iRet;                                           OUT
BYTE bErrorSelect;                                  IN
BYTE bErrorEject;                                   IN
BYTE bStamp;                                        IN
BYTE bCancel;                                       IN
DWORD dwTargetColor;                                IN
short sResolution;                                  IN
DWORD dwInfoMode;                                   IN
BARCODE_INFO stInfo(5);                             IN
      ypedef struct {
          int iRet;                    (OUT)
          BYTE bStatus;                (OUT)
          BYTE bDetail;                (OUT)
          DWORD dwSymbolMask;          (IN)
          BYTE bDirection;             (IN)
          BYTE bOrigin;                (IN)
          WORD wStartX;                (IN)
          WORD wStartY;                (IN)
          WORD wEndX;                  (IN)
          WORD wEndY;                  (IN)
      } BARCODE_INFO, *LPBARCODE_INFO;
BYTE bDataCount;                                    OUT
LPVOID lpData;                                      OUT
} MF_BARCODE, *LPMF_BARCODE;
```

- int iSize; (IN)
  This is the size of this structure.

- int iVersion; (IN)
  This is the structure version. Always specify MF_BARCODE_VERSION.
  Since the MF_BARCODE_VERSION value is different for each driver version, the application must always use the supplied header file.

- int iRet; (OUT)
  The execution result on barcode is set.

- BYTE bErrorSelect;(IN)
  Specifies whether to detect the barcode decode error.
  The valid commands are listed below.

Constant	Description
MF_ERROR_SELECT_NODETECT	No error detected
MF_ERROR_SELECT_DETECT	Error detected

  The default value is MF_ERROR_SELECT_DETECT.
  If MF_ERROR_SELECT_DETEC is specified, the settings of bErrorEject, bStamp, bCancel are ignored.

❏ BYTE bErrorEject;(IN)
Specifies where to eject when the barcode decode error is detected.
The valid commands are listed below.

Constant	Description
MF_EJECT_MAIN_POCKET	Ejected to the main pocket
MF_EJECT_SUB_POCKET	Ejected to the sub pocket
MF_EJECT_NOEJECT	Not ejected (completed with error)

The default value is MF_EJECT_MAIN_POCKET.
If MF_ERROR_SELECT_NODETECT is set to bErrorSelect, the setting of bErrorEject is ignored.
In the Confirmation mode, this setting is applied when BiSetBehaviorToScnResult cannot be called back in the MF_DATARECEIVE_DONE callback process. Reading speed decreases if bActivationMode of MF_PROCESS structure is set, and this setting is set to other than MF_EJECT_MAIN_POCKET.

❏ BYTE bStamp;(IN)
Specifies whether to do the flanking process to the sheet when the barcode decode error is detected. The valid commands are listed below.

Constant	Description
MF_STAMP_ENABLE	Franker eabled
MF_STAMP_DISABLE	Franker disabled

The default value is MF_STAMP_DISABLE.
If MF_ERROR_SELECT_NODETECT is set to bErrorSelect, the setting of bStamp is ignored.
In the High Speed mode, operates franker in accordance with this setting value.
In the Confirmation mode, if BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, operates franker in accordance with this setting value.
If the setting for bSuccessStamp of MF_PROCESS structure differs from this setting when reading is processed in the High Speed mode, reading speed slows down.

❏ BYTE bCancel; (IN)
Specifies whether to continue the reading process when the barcode decode error is detected. The valid commands are listed below.

Constant	Description
MF_CANCEL_DISABLE	Does not cancel the reading process for the next check paper.
MF_CANCEL_ENABLE	Cancels the reading process for the next check paper.

The default value is MF_CANCEL_DISABLE.
When MF_ERROR_SELECT_NODETECT is set to bErrorSelect, the setting of bCancel is ignored.
In the High Speed mode, operates in accordance with this setting value.
In the Confirmation mode, the insertion direction incorrect error is notified with MF_ERROR_OCCURRED callback and the reading process is continued. If BiSetBehaviorToScnResult is not invoked in the MF_DATARECEIVE_DONE callback notification, operates in accordance with this setting value.
When reading is processed in the High Speed mode, if this setting is set to other than MF_CANCEL_ENABLE, reading speed slows down.

❏ DWORD dwTargetColor; (IN)
Specifies the barcode color.
Specify BARCODE_TARGET_COLOR_GRAYSCALE.

❏ short sResolution; (IN)
Specifies the image resolution of image data.
Specify MF_SCAN_DPI_DEFAULT (200 dpi).

❏ DWORD dwInfoMode; (IN)
Specify 0.

❏ BARCODE_INFO stInfo[5]; (IN)
This is the structure where the decode setting and the status of decode result are made. Five arrangements are available.

- int iRet; (OUT)
  The barcode decode execution result is set.
  For the details, refer to "MF_BARCODE.iRet" on page 4-64.

- BYTE bStatus; (OUT)
  The status of decoding result is set.

Bit	Function	Value	
		0	1
0,1,2	Reserved	Fixed to 0	
3	Detailed information	-	Added
4	Reserved	Fixed to 0	
5	Reading result	Success	Failure
6,7	Reserved	Fixed to 0	

- BYTE bDetail; (OUT)
  The detailed status of decoding result is set.

Value	Information
40h	Success
45h	Barcode cannot be detected.

- DWORD dwSymbolMask; (IN)
  Specifies the type of barcode symbol to decode. Make sure to set this because the default value is not set for this setting. A combination of multiple barcode symbols can be set.

Constant	Barcode
MF_BARCODE_SYMBOL_CODABAR	Codabar
MF_BARCODE_SYMBOL_CODE128	Code128
MF_BARCODE_SYMBOL_CODE39	Code39
MF_BARCODE_SYMBOL_ITF	ITF
MF_BARCODE_SYMBOL_EAN_JAN	JAN13(EAN), JAN8(EAN)
MF_BARCODE_SYMBOL_UPC_A	UPC-A
MF_BARCODE_SYMBOL_UPC_E	UPC-E

  <Example: When specifying Codabar and ITF>

```
MF_BARCODE.stInfo[0].dwSymbolMask = (MF_BARCODE_SYMBOL_CODABAR |
                                     MF_BARCODE_SYMBOL_ITF);
```

- BYTE bDirection; (IN)
  Sets the barcode decode direction.
  The default value is MF_BARCODE_DIRECTION_ALL.

Constant	Description
MF_BARCODE_DIRECTION_ALL	Decodes it in both directions from left to right and top to bottom
MF_BARCODE_DIRECTION_LEFTRIGHT	Decodes it from left to right
MF_BARCODE_DIRECTION_TOPBOTTOM	Decodes it from top down
MF_BARCODE_DIRECTION_RIGHTLEFT	Decodes it from right to left
MF_BARCODE_DIRECTION_BOTTOMTOP	Decodes it from bottom up

- BYTE bOrigin; (IN)
  Setting is not necessary.

- WORD wStartX; (IN)
  Setting is not necessary.

- WORD wStartY; (IN)
  Setting is not necessary.

- WORD wEndX; (IN)
  Setting is not necessary.

- WORD wEndY; (IN)
  Setting is not necessary.

❑ BYTE bDataCount; (OUT)
The number of scanned barcodes is set.

❏ LPVOID lpData; (OUT)
The result of barcode decoding (BARCODE_DATA structure) is set.
The BARCODE_DATA structure is an element of arrangement. The number of elements is the number of read barcodes (bDataCount). If there is no decode result of barcode, NULL is set.

<BARCODE_DATA structure>

```
typedef struct {
    DWORD dwSymbol;          OUT
    BYTE bDirection;         OUT
    WORD wStartX;            OUT
    WORD wStartY;            OUT
    WORD wEndX;              OUT
    WORD wEndY;              OUT
    DWORD dwDataSize;        OUT
    LPVOID pData;            OUT
} BARCODE_DATA, *LPBARCODE_DATA;
```

- DWORD dwSymbol; (OUT)
  The scanned barcode symbol is set.
  Refer to dwSymbolMask (4-179) for the set value.

- BYTE bDirection; (OUT)
  The direction of decoded barcode is set.

Constant	Description
MF_BARCODE_DIRECTION_LEFTRIGHT	Decodes it from left to right
MF_BARCODE_DIRECTION_TOPBOTTOM	Decodes it from top down
MF_BARCODE_DIRECTION_RIGHTLEFT	Decodes it from right to left
MF_BARCODE_DIRECTION_BOTTOMTOP	Decodes it from bottom up

- WORD wStartX; (OUT)
  The starting of decoding in the x-coordinate (dot unit) of scanned barcode is set.

- WORD wStartY; (OUT)
  The starting of decoding in the y-coordinate (dot unit) of scanned barcode is set.

- WORD wEndX; (OUT)
  The end of decoding in the x-coordinate (dot unit) of scanned barcode is set.

- WORD wEndY; (OUT)
  The end of decoding in the y-coordinate (dot unit) of scanned barcode is set.

- DWORD dwDataSize; (OUT)
  The size of the scanned barcode's binary data is set.
  The size including the NULL at the end is set if the set data(pData) is a character string.

- LPVOID pData; (OUT)
  The scanned barcode's binary data is set.
  The size including the NULL at the end is set if the set result is a character string.
  Free the reference memory with GlobalFree.

Chapter 5
Differences Between TM-J9000/J9100 API and TM-S1000 API

This chapter describes the differences between the TM-S1000 API and the TM-J9000/J9100 API to enable you to use an application for the TM-J9000/J9100 with one for the TM-S1000.

The differences between the APIs for the TM-J9000/J9100 and the TM-S1000 are as follows:

- ❏ The TM-J9000/J9100 is a printer with a scanner/photo ID function, while the TM-S1000 is a device for scanning only. The TM-J9000/J9100 APIs for printer/Photo ID are not available for the TM-S1000. If they are used, a value of ERR_NOT_SUPPORT will be returned.
- ❏ Even among the TM-J9000/J9100 APIs available for the TM-S1000, there are ones whose parameters specified, return value, or event content are different. If any parameter that is not supported is specified, a value of ERR_NOT_SUPPORT will be returned.

API Lists of the TM-J9000/J9100 and the TM-S1000

Consider the following information for your applications.

Compatibility Between the TM-J9000/J9100 and the TM-S1000

New: API newly added for the TM-S1000. It is not available for the TM-J9000/J91000.

Up: API whose function was extended from the TM-J9000/J9100 API. It is upward compatible. It enables efficient use of the TM-S1000 functions.

OK: API that is compatible with the TM-S1000 API. Use as it is.

Change: API that has the same name as that of the TM-J9000/J9100 API, but that has a different input or output. Refer to the differences to change the source and use it.

NG: API only for the TM-J9000/J9100. It is not supported by the TM-S1000.

API	Compatibility	Different information				
		Input	Output	Description	Item	Data
BiOpenMonPrinter	***Change***	✔	-	Specified product name is different.	pNAME	TYPE_PORT : USB2 TYPE_PRINTER : TM-S1000U
BiSetMonInterval	***OK***	-				
BiGetStatus	***Change***	-	✔	Obtained Devie status information is different.	lpStatus	See "Device Status" on page 4-1
BiSetStatusBackFunction	***OK***	-				
BiSetStatusBackWnd	***OK***	-				
BiCancelStatusBack	***OK***	-				

API	Compatibility	Different information				
		Input	Output	Description	Item	Data
BiDirectIO	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiDirectIOEx	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiResetPrinter	***OK***	-				
BiGetCounter	***Change***	✔	✔	Specifying a maintenance counter that is not present leads to an error. (ERR_PARAM)	readno	See "Maintenance Counter" on page 4-3
BiResetCounter	***Change***	✔	-	Specifying a maintenance counter that is not present leads to an error. (ERR_PARAM)	writeno	See "Maintenance Counter" on page 4-3
BiCancelError	***OK***	-				
BiGetType	***Change***	-	✔	Obtained device information is different.	typeid font exrom euspecial	See "Type ID" on page 4-3
BiGetOfflineCode	***Change***	-	✔	Obtained offline information is different.	offlinecode	See "Offline Code (BiGetOfflineCode)" on page 4-6
BiGetOfflineCodeByIndex	***New***	-		Obtains offline information by bytes.		
BiGetInkStatus	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiSetInkStatusBackFunction	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiSetInkStatusBackWnd	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiCancelInkStatusBack	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiMICRSelectDataHandling	***OK***	-		When MF_PROCESS structure is not set, the setting is enabled.		
BiMICRGetStatus	***Change***	-	✔	Diifferent MICR information is obtained.	pStatus	See "MICR Status" on page 4-10
BiMICRCleaning	***OK***	-				
BiSCNSetImageQuality	***OK***	-				
BiSCNSetImageFormat	***OK***	-				
BiSCNSetScanArea	***OK***	-				
BiSCNGetImageQuality	***OK***	-				

API	Compatibility	Different information				
		Input	Output	Description	Item	Data
BiSCNGetImageFormat	***OK***	-				
BiSCNGetScanArea	***OK***	-				
BiSCNGetClumpStatus	***NG***	-				
BiSCNSetCroppingArea	***OK***	-				
BiSCNGetCroppingArea	***OK***	-				
BiSCNDeleteCroppingArea	***OK***	-				
BiSCNSelectScanUnit	***Change***	✔	-	Specifying Card leads to an error. (ERR_PARAM)	bSelectUnit	Only BPS_BI_SCN_UNIT_CHECKPAPER is specified.
BiSCNMICRFunction	***Up***	✔	-	MF_PROCESS structure was added to the input information. When the operation is set with BiMICRSelectDataHandling, reading is possible even if it is not set.	MF_PROCESS	See "Setting List of MF_PROCESS Structure" on page 5-6
BiSCNMICRCancelFunction	***OK***	-				
BiSCNSelectScanFace	***OK***	-				
BiGetPrnCapability	***Change***	✔	✔	Obtained device information is diffrent.	pmID	See "Device ID" on page 4-4
BiCloseMonPrinter	***OK***	-				
BiGetRealStatus	***Change***	-	✔	Obtained Devie status information is different.	lpStatus	See "Device Status" on page 4-1
BiSendDataFile	***NG***	-				
BiDirectSendRead	***NG***	-				
BiSetStatusBackFunctionEx	***OK***	-				
BiSetInkStatusBackFunctionEx	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiSCNMICRFunctionContinuously	***Up***	✔	-	MF_PROCESS structure was added to the input information. When the operation is set with BiMICRSelectDataHandling, reading is possible even if it is not set.	MF_PROCESS	See "Setting List of MF_PROCESS Structure" on page 5-6

API	Compatibility	Different information				
		Input	Output	Description	Item	Data
BiSCNMICRFunctionPostPrint	***Up***	✔	-	MF_PROCESS structure was added to the input information. When the operation is set with BiMICRSelectDataHandling, reading is possible even if it is not set.	MF_PROCESS	See "Setting List of MF_PROCESS Structure" on page 5-6
BiSCNMICRSetStatusBackFunction	***OK***	-				
BiSCNMICRSetStatusBackWnd	***OK***	-				
BiSCNMICRCancelStatusBack	***OK***	-				
BiSetNumberOfDocuments	***New***	-				
BiGetMicrText	***OK***	-				
BiMICRClearSpaces	***New***	-				
BiSetOcrABAreaOrigin	***New***	-				
BiGetOcrABText	***New***	-				
BiGetScanImage	***OK***	-				
BiGetBarcodeData	***New***	-				
BiDecodeBarcode	***New***	-				
BiDecodeBarcodeMemory	***New***	-				
BiGetTransactionNumber	***OK***	-				
BiSetTransactionNumber	***OK***	-				
BiGetPrintStation	***OK***	-				
BiSetPrintStation	***Change***	✔	-	Specifying Receipt or Validation leads to an error. (ERR_PARAM)	wStation	
BiPrintText	***OK***	-				
BiPrintBarCode	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiPrintImage	***OK***	-				
BiPrintMemoryImage	***New***	-				
BiGetPrintAlignment	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiSetPrintAlignment	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiGetPrintSize	***OK***	-				
BiSetPrintSize	***OK***	-				
BiGetPrintPosition	***OK***	-				
BiSetPrintPosition	***OK***	-				
BiSetEndorseDirection	***New***	-				
BiUpdateEndorseText	***OK***	-				

API	Compatibility	Different information				
		Input	Output	Description	Item	Data
BiInsertValidation	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiRemoveValidation	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiBufferedPrint	***OK***	-				
BiGetPrintControl	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiSetPrintControl	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiSetTransactionNumberWithIncremental	***OK***	-				
BiSelectJamDetect	***NG***	-		Leads to an error. (ERR_NOT_SUPPORT)		
BiSetBehaviorToScnResult	***New***	-				
BiSetPaperThickness	***New***	-				
BiRingBuzzer	***New***	-				
BiSetWaterfallMode	***New***	-				
BiGetIQAResult	***New***	-				
BiGetVersion	***New***	-		API to obtain version of the driver and module in use.		

API Compatibility of the Scanner Extended Functions

API of scanner extended functions	Compatibility
BiESCNEnable	***OK***
BiESCNGetAutoSize	***OK***
BiESCNSetAutoSize	***OK***
BiESCNGetCutSize	***OK***
BiESCNSetCutSize	***OK***
BiESCNGetRotate	***OK***
BiESCNSetRotate	***OK***
BiESCNGetDocumentSize	***OK***
BiESCNSetDocumentSize	***OK***
BiESCNGetDeSkew	***New***
BiESCNSetDeSkew	***New***
BiESCNDefineCropArea	***OK***
BiESCNGetMaxCropAreas	***OK***
BiESCNStoreImage	***OK***
BiESCNRetrieveImage	***OK***
BiESCNClearImage	***OK***
BiESCNGetRemainingImages	***OK***

Setting List of MF_PROCESS Structure

Member	Data type	Description
iSize	int	Specifies the structure size.
iVersion	int	Specifies the structure version.
bActivationMode	BYTE	Specifies the process mode (error judging method).
bPaperType	BYTE	Specifies the type of a check to be scanned.
dwStartWaitTime	DWORD	Specifies the waiting time for insertion.
bSuccessStamp	BYTE	Specifies the franking process when MICR reading is successful.
bPaperMisInsertionErrorSelect	BYTE	Sets the error judgement when the insertion orientaion is wrong.
bPaperMisInsertionErrorEject	BYTE	Sets the ejection method when an insertion orientation error occurs.
bPaperMisInsertionStamp	BYTE	Sets the franking process when an insertion orientation error occurs.
bPaperMisInsertionCancel	BYTE	Sets cancellation when an insertion orientation error occurs.
bDoubleFeedErrorSelect	BYTE	Sets the error judgement when a double-feeding error occurs.
bDoubleFeedErrorEject	BYTE	Sets the ejection method when a double-feeding error occurs.
bDoubleFeedStamp	BYTE	Sets the franking process when a double-feeding error occurs.
bDoubleFeedCancel	BYTE	Sets the cancellation when a double-feeding error occurs.
bBaddataErrorSelect	BYTE	Sets the error judgement when an unrecognized MICR character is detected.

Member	Data type	Description
bBaddataCount	BYTE	Sets the allowed number of characters when an unrecognized MICR character error ocurs.
bBaddataErrorEject	BYTE	Sets the ejection method when an unrecognized MICR character error occurs or when the number of unrecognized MICR characters exceeds the allowance.
bBaddataStamp	BYTE	Sets the franking process when an unrecognized MICR character error occurs or when the number of unrecognized MICR characters exceeds the allowance.
bBaddataCancel	BYTE	Sets the cancellation when an unrecognized MICR character error occurs or when the number of unrecognized MICR characters exceeds the allowance.
bNodataErrorSelect	BYTE	Sets the error judgement when the MICR magnetic waveform is not detected.
bNodataErrorEject	BYTE	Sets the ejection method for a MICR magnetic waveform undetection error.
bNodataStamp	BYTE	Sets the franking process when a MICR magnetic waveform undetection error occurs.
bNodataCancel	BYTE	Sets the cancellation when a MICR magnetic waveform undetection error occurs.
bNearFullSelect	BYTE	Sets the near-full pocket detection.
bResultPartialData	BYTE	Sets the data handling when an error occurs during scanning.

Setting List of MF_IQA Structure

Member	Data type	Description
iSize	int	Specifies the structure size.
iVersion	int	Specifies the structure version.
bErrorSelect	BYTE	Enables/Disables the IQA function.
bErrorEject	BYTE	Specifies the ejection pocket of documents.
bStamp	BYTE	Specifies the franking process.
bCancel	BYTE	Specifies whether to continue or cancel the reading process of a document.
bImageFormat	BYTE	Specifies an image format.
bColorDepth	BYTE	Specifies the gradation.
bThreshold	CHAR	Specifies the density threshold.
bColor	BYTE	Specifies color.
bExOption	BYTE	Specifies the variety of density adjustment.
sResolution	short	Specifies the resolution.
bUndersize	BYTE	Enables/Disables UndersizeImage validation.
bOversize	BYTE	Enables/Disables OversizeImage validation.
bMincompressed	BYTE	Enables/Disables MinCompressedImageSize validation.
bMaxcompressed	BYTE	Enables/Disables MaxCompressedImageSize validation.

Member	Data type	Description
bFront_rear	BYTE	Enables/Disables FrontRearImageMismatch validation.
bToolight	BYTE	Enables/Disables ImageTooLight validation.
bToodark	BYTE	Enables/Disables ImageTooDark validation.
bStreaks	BYTE	Enables/Disables HorizontalStreaksPresent validation.
bNoise	BYTE	Enables/Disables ExcessiveSpotNoise validation.
bFocus	BYTE	Enables/Disables ImageOutOfFocus validation.
bCorners	BYTE	Enables/Disables ImageOutOfFocus validation.
bEdges	BYTE	Enables/Disables FoldedTornDocEdges validation.
bFraming	BYTE	Enables/Disables DocFramingError validation.
bSkew	BYTE	Enables/Disables ExcessiveDocSkew validation.
bCarbon	BYTE	Enables/Disables CarbonStripDetection validation.
bPiggyback	BYTE	Enables/Disables Piggyback validation.

Setting List of MF_BARCODE Structure

Member	Data type	Description
iSize	int	Specifies the structure size.
iVersion	int	Specifies the structure version.
iRet	int	The execution result on barcode is set.
bErrorSelect	BYTE	Specifies whether to detect the barcode decode error or not.
bErrorEject	BYTE	Specifies where to eject when the barcode decode error is detected.
bStamp	BYTE	Specifies whether to do the flanking process to the sheet when the barcode decode error is detected.
bCancel	BYTE	Specifies whether to continue read processing when the barcode decode error is detected.
dwTargetColor	DWORD	Specifies the barcode color.
sResolution	short	Specifies the image resolution of image data.
dwInfoMode	DWORD	Fixed to 0
stInfo(5)	BARCODE_INFO	The structure where the decode setting and the status of decode result are made.
stInfo().iRet	int	The decode execution result is set.
stInfo().bStatus;	BYTE	The decoding result status is set.
stInfo().bDetail;	BYTE	The detailed status of decoding result is set.
stInfo().dwSymbolMask;	DWORD	Specifies the type of barcode symbol to decode
stInfo().bDirection;	BYTE	Specifies the direction to decode.
stInfo().bOrigin;	BYTE	Specifies the origin point to decode.
stInfo().wStartX;	WORD	Specifies the starting point to decode in the x-coordinate.
stInfo().wStartY;	WORD	Specifies the starting point to decode in the y-coordinate.
stInfo().wEndX	WORD	Specifies the ending point to decode in the x-coordinate.
stInfo().wEndY	WORD	Specifies the ending point to decode in the y-coordinate.
bDataCount	BYTE	The number of barcode decoding results is set.
lpData	LPVOID	The barcode decoding result (BARCODE_DATA structure) is set.

Chapter 6
Log Collection Function

This chapter describes how to create a log file and how to analyze it.

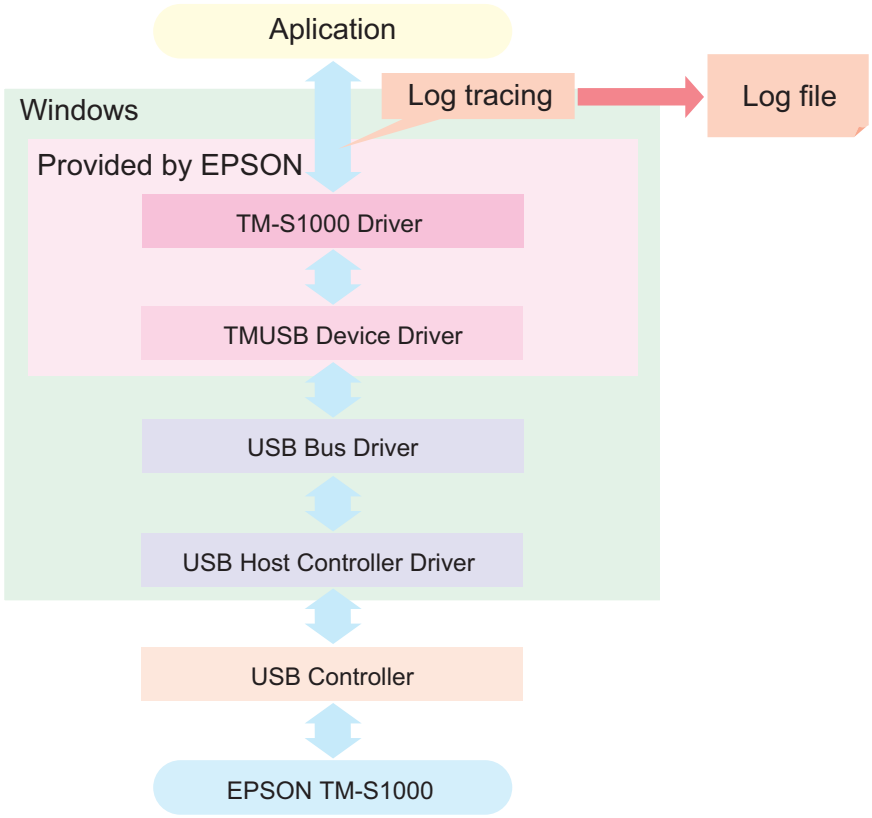
A log file records tracing between an application and an API. (File name: TMS1000DriverTrace.log) The log file records TM-S1000 APIs executed, parameters, obtained MICR/OCR data and so on. Log files are helpful for efficient application development and error analysis.

Note:
Image data and its file name are not recorded.
The recordable file size is up to 10MB. If a file size exceeds 10MB, the file name (TM-S1000DriverTrace_Bak.log) will be changed and a new log file will be created.

Log files are stored in the following folder depending on the OS.

- Windows 2000 / Windows XP
  C:\Documents and Settings\All Users\EPSON\BANK\TM-S1000\Trace
- Windows Vista or newer Windows versions.
  C:\ProgramData\EPSON\BANK\TM-S1000\Trace

The following description is for Windows XP Professional.



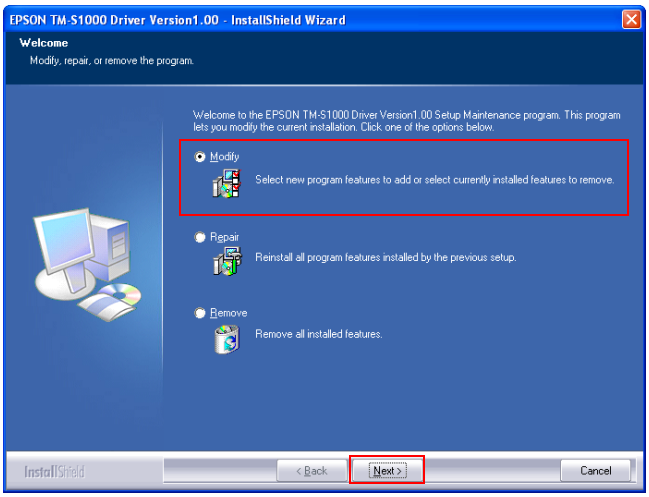
Creating A Log File

Follow the steps below to create a log file or to quit creating a log file.

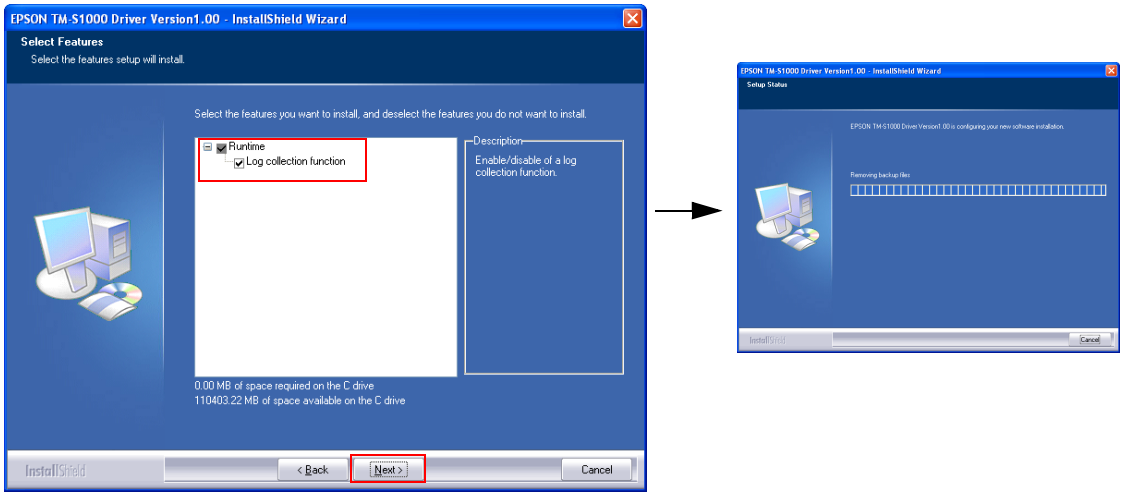
How to Create A Log File

Use setup.exe of the TM-S1000 Driver to create a log file. Follow the steps below for setting.

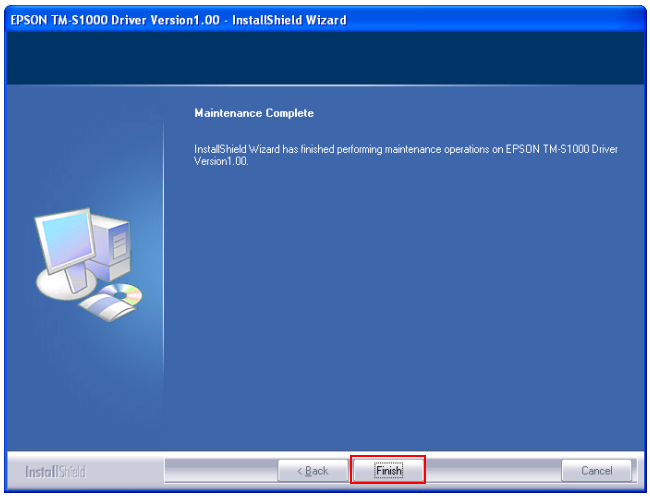
1. When the TM-S1000 Driver is installed, start up setup.exe of EPSON TM-S1000 Driver.
2. Select [Modify], and then click [Next].



3. The “Select Features” screen appears. Check the checkbox for [RunTime] and [Log collection function], and then click [Next].



4. The “Maintenance Complete” screen appears. Click [Finish] to exit.

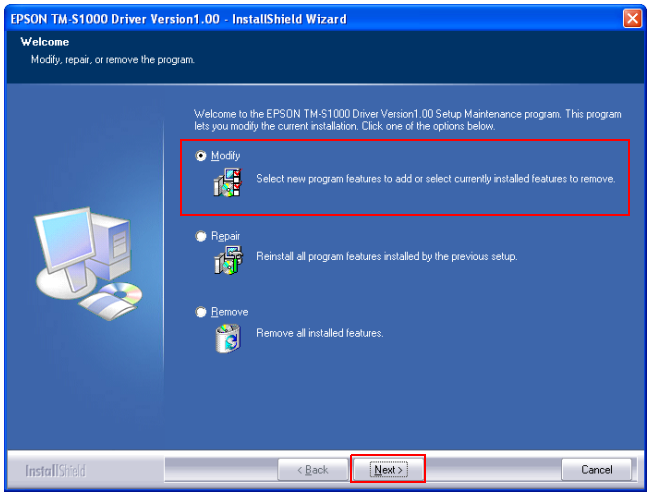


Hereafter, a log file will be created when you use the TM-S1000 API.

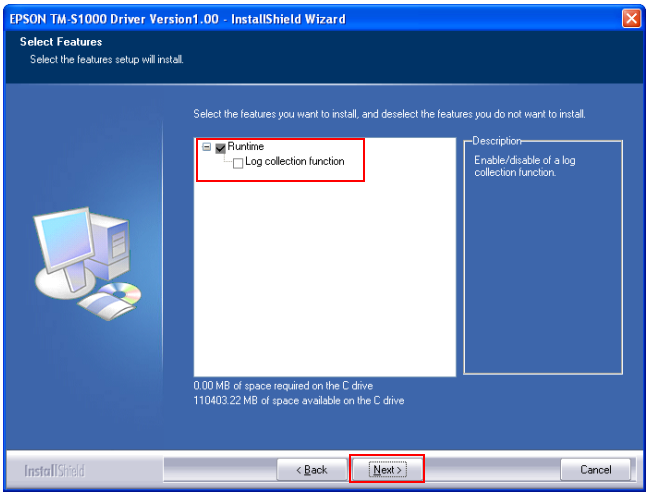
How to Quit Creating A Log File

Use setup.exe of the TM-S1000 Driver also to quit creating a log file.

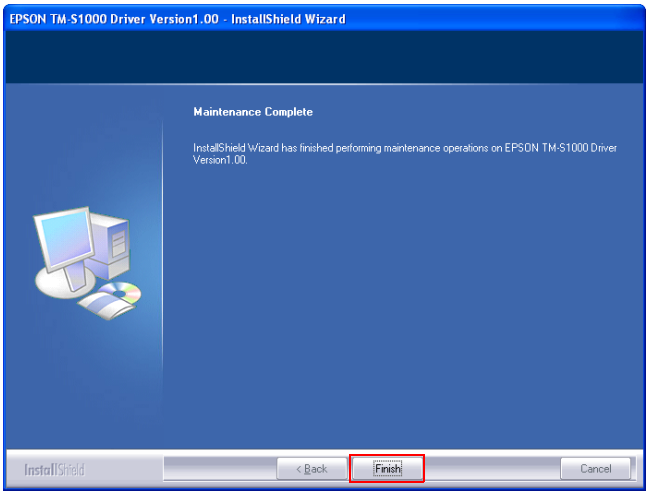
1. When the TM-S1000 Driver is installed, start up setup.exe of EPSON TM-S1000 Driver.
2. Select [Modify], and then click [Next].



3. The “Select Features” screen appears. Uncheck the check box for [Log collection function] under [RunTime].



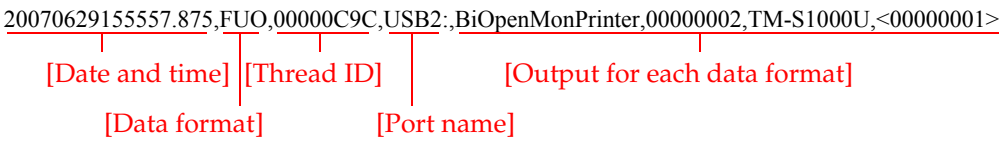
4. The “Maintenance Complete” screen appears. Click [Finish] to exit.



Hereafter, a log file will not be created even when you use the TM-S1000 API.

How to Analyze A Log File

One record of a log file shows the following information:



Common output

Contents	Output date and time "YYYYMMDDhhmmss.sss"	,	Data format "FUI" : Calling API "CBI" : Calling CALLBACK function "FUO" : Return of API "CBO" : Return of CALLBACK function "MFB" : Driver accesses the OUT attribute member of MF_BASE01 structure the application has. "MFM" : Driver accesses the OUT attribute member of MF_MICR structure. "MFS" : Driver accesses the OUT attribute member of MF_SCAN structure. "MFP" : Driver accesses the OUT attribute member of MF_PRINT01 structure. "ASB" : ASB notification from the device "SRC" : Event that is notified to CALLBACK during reading operation.	,	Thread ID	,	Port name	,
Number of columns	18	1	3	1	8	1	3	1

Details of data format

Data format	Output
"FUI"	"API name, parameter value 1,..., parameter value n" If there is more than one parameter, they continue in the specified order using "," as a separator. If a parameter is a structure, the log file is broken into members (1 parameter = 1 member). The start of a structure and array is shown as "{" and the end of them is shown as "}." Parameter values are output in a decimal or hexadecimal character string.
"CBI"	"Handle, parameter value 1,..., parameter value n" If there is more than one parameter, they continue in the specified order using "," as a separator. Parameter values are output in a decimal or hexadecimal character string.
"FUO"	"API name, parameter value 1,..., parameter value n, <return value>" If there is more than one parameter, they continue in the specified order using "," as a separator. If a parameter is a structure, the log file is broken into members (1 parameter = 1 member). The start of a structure and array is shown as "{" and the end of them is shown as "}." Parameter values are output in a decimal or hexadecimal character string. The value shown in the last <> shows a return value. The return value is shown in a hexadecimal character string.

Data format	Output
"CBO"	"Handle, parameter value 1,..., parameter value n, <return value>" If there is more than one parameter, they continue in the specified order using "," as a separator. If a parameter is a structure, the log file is broken into members (1 parameter = 1 member). Parameter values are output in a decimal or hexadecimal character string. The value shown in the last <> shows a return value. The return value is shown in a hexadecimal character string.
"MFB"	"Handle, contents of MF_BASE01 structure..." If a parameter is a structure, the log file is broken into members (1 parameter = 1 member). The start of a structure and array is shown as "{" and the end of them is shown as "}." Parameter values are output in a decimal or hexadecimal character string.
"MFM"	"Handle, contents of MF_MICR structure..." If a parameter is a structure, the log file is broken into members (1 parameter = 1 member). The start of a structure and array is shown as "{" and the end of them is shown as "}." Parameter values are output in a decimal or hexadecimal character string.
"MFS"	"Handle, contents of MF_SCAN structure..." If a parameter is a structure, the log file is broken into members (1 parameter = 1 member). The start of a structure and array is shown as "{" and the end of them is shown as "}." Parameter values are output in a decimal or hexadecimal character string.
"MFP"	"Handle, contents of MF_PRINT01 structure..." If a parameter is a structure, the log file is broken into members (1 parameter = 1 member). The start of a structure and array is shown as "{" and the end of them is shown as "}." Parameter values are output in a decimal or hexadecimal character string.
"ASB"	"Handle, ASB notified from the device" Values are output in a hexadecimal character string.
"SRC"	Outputs differ depending on the events. Values are output in a hexadecimal character string. Reading start Event type: 1 "Handle, event type, transaction number, reading result" Reading result Event type: 2 "Handle, event type, transaction number, reading result" Data reception Event type: 3 "Handle, transaction number, reading result, analysis result of magnetic waveform data, analysis result of OCR, reception result of image data" Ejection end Event type: 4 "Handle, event type, transaction number, reading result" Reading end Event type: 5 "Handle, event type received, reading result"

Examples of log file output

```
20070703164335.421,FUI,00000854,    :,BiESCNEnable,00000001
20070703164335.421,FUO,00000854,    :,BiESCNEnable,00000001,<00000000>
20070703164335.421,FUI,00000854,    :,BiOpenMonPrinter,00000002,TM-S1000U,1.00
20070703164336.156,ASB,000000AC,USB2:,00000001,4D6F0014
20070703164336.156,FUO,00000854,USB2:,BiOpenMonPrinter,00000002,TM-S1000U,<00000001>
20070703164336.156,FUI,00000854,USB2:,BiSCNMICRSetStatusBackFunction,00000001,004DD3FF
20070703164336.156,FUO,00000854,USB2:,BiSCNMICRSetStatusBackFunction,00000001,004DD3FF
,<00000000>
20070703164336.203,FUI,00000854,USB2:,BiSCNMICRFunctionPostPrint,00000001,{00000000,0000
0000,00000000,00000000,{00000000,00000000},00000000,00000000,{00000000,00000000,00000000}
,{00000000,00000000,00000000},00000000,00000000},00000030
20070703164336.203,FUO,00000854,USB2:,BiSCNMICRFunctionPostPrint,00000001,{00000127,0000
0101,CCCCCCCC,00000003,00000000,{00000000,00000000},00000000,00000001,{00000000,000000
00,00000000},{00000000,00000000,00000000},00000000,USB2,00000000},00000010,<00000000>
20070703164336.203,FUI,00000854,USB2:,BiSCNMICRFunctionPostPrint,00000001,{00000000,0000
0000,00000000,00000000,00000000,NULL},00000032
```